Worksheet 1: Starting with Prolog

Due: Midnight, Monday, 24 January 2010.
Total marks: 25+3

Instructions

If you detect or suspect any errors, post a question on the Worksheet 1 discussion forum for clarification, or email the instructor!

The Worksheet should be solvable in 6 to 8 hours of work. If after two hours (not including reading the text and notes), you have not made significant progress, waste no further time, and consult with the TA or the instructor.

Submit your solutions electronically using the CMPT 340 Moodle page for Worksheet 1. Please allow yourself some extra time to upload and submit your work, to account for using a system that may be new to you.

Question One (7 marks)

In this question, you will write a few simple Prolog facts and rules to practice expressing relationships, and using terms as structured data.

The problem domain is a calendar application. In a calendar app, you record events, which have a date, a start time, an end time, and a few details about what the event is about. For our purposes, the details will be be unimportant, even though in a real application, they are the whole point. The application will assume that events are restricted to a single day.

The file `sched.pl` is given on the Worksheet 1 link. In it you will find comments and documentation about this question. In particular, you will note that there is a large section of comments that describe 4 data structures (terms). These are:

- `time24/2` Represents a time using a 24 hour clock (hours, minutes only).
- `date/3` Represents a date using day, month, year
- `event/4` Represents an event using a date, start time, end time, and a place for any other detail (say, an event name, or other information).

Full details about the representation are in the file, and are not reproduced here.

Your task will be to write a few simple Prolog predicates for these data structures. These parts are also fairly well documented in the file.

1. Write down your class schedule for this term, using the `anEvent/1` predicate.
2. Write a predicate called `before_time/2` which takes 2 times, and succeeds if the first time is before the second. Assume that the two given times are on the same day. You should use this predicate in the remaining predicates, below.
3. Write a predicate called `meet/2` that takes 2 events as arguments, and succeeds if the first event ends exactly when the second begins. Assume that the 2 events are on the same day.

4. Write a predicate called `during/2` that takes 2 events as arguments, and succeeds if the first event happens during the second. In other words, the first event starts after the second event starts, and ends before the second one ends. (Revised) Assume that the 2 events are on the same day.

5. Write a predicate called `before/2` that takes 2 events as arguments, and succeeds if the first event ends before the second event starts. Assume that the 2 events are on the same day.

6. Write a predicate called `overlap/2` that takes 2 events as arguments, and succeeds if the two events overlap, i.e., there is some period of time that both events are going on. (I am being a bit vague here, deliberately so that you have to make the concept more precise, before you implement it. It's not intended to be tricky, though.) Assume that the 2 events are on the same day.

Evaluation

Marking will be based on the following attributes:

- Correct implementation: 1 mark each. No part marks. The mark will be deducted if the format of your program is unacceptable, even if it is correct.
- The documentation is mostly done for you. But if you add any significant predicates, they must be documented.

What to hand in

Your implementation of the predicates above, in a file called `w1q1.pl`. Make sure that you put student information in the file, and document anything that you might think will help the marker read your program.

Question Two (8 marks)

Researchers have developed a "hyper-space drive" which will enable interstellar travel across the universe in exceedingly short amounts of time. It works as follows.

Any distance of 1 meter or less can be travelled in one minute (this is slow, but remember, these vehicles are very large, and moving faster would be dangerous). For any distance greater than one meter, the distance to the destination is determined, and then the drive is set to jump exactly half of that distance. Every jump takes exactly one minute. As you can see, the drive can make very long jumps at first, but the size of the jump is cut in half every minute.[1]

For example, if the ship has to travel 10 meters, it will travel 5 meters in the first minute, 2.5 meters the second minute, 1.25 meters the 3rd minute, and 0.625 meters the fourth minute. Finally, the remaining 0.625 meters takes one more minute. Thus the total time to travel 10 meters is 5 minutes.

Write two recursive Prolog programs to determine how many minutes it will take to travel any distance. Assume distances are always positive, and measured in meters.

[1] This is an exceptionally fast ship. It takes about 30 minutes to travel the distance between the Earth and our Sun, and less than an hour to travel to the next nearest star. It has not been tested on the Kessel run. For obvious reasons, the drive is called "The Zeno Drive."

- A simple recursive version, based on the factorial code we have seen in class, has time and space complexity $O(N)$.
- A "tail recursive" version that is $O(N)$ time and $O(1)$ space.

This problem is deliberately simple so that you can take the time to think about it "declaratively." Think about what relationship you are describing, rather than just "how to get the right answer." Of course, you have to think of both, but pay extra attention addressing the truth of the relationships you are describing.

Evaluation

Marking will be based on the following attributes:

- Three marks (each): Correct implementation. One mark will be deducted if the format of your program is unacceptable, even if the program is correct.
- One mark (each): Documentation: your program should be documented with an informal contract, as discussed in the notes and in class. A long discussion about the algorithm is unnecessary.

What to hand in

Two correctly working programs, in a file called `w1q2.pl`.

Question Three (10 marks)

Prolog has integer division and remainder built-in:

```
?- Q is 18//4.
Q = 4 ?
yes
?- R is 18 mod 4.
R = 2 ?
yes
```

We will pretend that it does not, just for fun and practice.

We will implement a Prolog program that computes both the integer quotient and remainder in the same program. Something like this:

```
?- divide(18,4,Q,R).
Q = 4
R = 2 ?
yes
```

In the above program, the first two arguments are the dividend and the divisor. The third argument is the integer quotient, and the last is the remainder.

The point of this exercise is to practice recursion. So we will restrict the program in the following way: the only arithmetic allowed is addition and subtraction. No division or multiplication. Don't even think about it.

Of course, it's not efficient, but we can divide by repeated subtraction. Just count the number of times you can subtract 4 from 18, without going into negative numbers. For further simplicity, we will assume that `divide` only has to work on natural numbers (no negative numbers).

Write 2 versions of `divide/4`:

1. A simple version, based on the factorial code we have seen in class, has time and space complexity $O(N)$.
2. A "tail recursive" version that is $O(N)$ time and $O(1)$ space.

It will help a lot if you reason about what your program should say declaratively. Think about what relationship you are describing, rather than "how to divide by subtraction." Of course, you have to think of both, but pay extra attention addressing the truth of the relationships you are describing. This problem is deliberately simple so that you can take the time to think about it this way.

Evaluation

Marking will be based on the following attributes:

- Four marks (each): Correct implementation. One mark will be deducted if the format of your program is unacceptable, even if the program is correct.
- One mark (each): Documentation. The documentation should include the informal contract, and a description of the variables and the relationship being defined, similar to what has been shown in class. You don't have to document every line, or write a half page describing what each step of your program does.

What to hand in

Two correctly working programs, in a file called `w1q3.pl`.

Bonus: Question Four (3 marks)

You will find a file called `mystero.pl` on the Moodle site (Week 3). Discover what it does. Format it, document it, and submit your formatted, documented version. Be sure to change the names of the predicates to something reasonably meaningful. The program uses material that we haven't studied yet. That shouldn't stop you. The point of this exercise is:

- To show by example why Prolog programs should be well-documented and well-formatted. This is exactly the exercise the marker has to go through if you do not follow documentation standards.
- To get you to reason about a program. There's almost no way to guess how to start it, or how to use it. You have to look for clues in the program, and you have to be prepared to test your hypotheses.

Evaluation

If your submission is reasonably well documented and well-formatted, you will receive 3 bonus marks.

What to hand in

A well-documented, well-formatted program, in a file called `w1q4.pl`.