Homework Assignment 3

Posted: 2/23/2010
Due: ~~3/16/2010~~ **extended to** 3/18/2010

The assignment is due in class; please follow the instructions on the homework submission form.

The coverage of Prolog was necessarily not exhaustive. You are expected to **study the Prolog resources made available on the class homepage** prior to asking for help.

I strongly recommend solving (most of) the assignment before spring break because there will be no office hours after spring break before the assignment is due. Make sure you have at least a solid idea on how to solve each part by 03/02/2010.

Objectives

Write basic Prolog clauses and queries.

Write recursive Prolog clauses.

Implement a recursive data structure in Prolog (extra credit).

Related Files

Homework submission form:
http://www.cs.unc.edu/Courses/comp524-s10/hw/submission-form.pdf

Part 1

Solve "Practical Exercise 2, part (1)" on page 27 of *Logic Programming with Prolog* [1].

Questions (a)–(d) ask you to devise and test goals. Copy and paste your queries and the obtained answers into a plain text file.

You do **not** have to solve part (2), which is concerned with persons.

Part 2

Write a knowledge base and rules for a simple security policy:

- Murray, Alan, and John are administrators.
- Brian is the director.
- Björn, Dawn, Donna, Katrina, and Tricia are users.
- Björn teaches COMP 524.
- Aaron teaches COMP 110.
- Any administrator may update any class mailing list.
- A director may act as an administrator at any time.
- A class mailing list may be be updated by the person that is teaching the class.
- A user may read any class mailing list.

Design and test queries that **a)** finds all people that may update COMP 524's class mailing list and **b)** find all people that may read COMP 524's class mailing list.

Part 3

Write a clause `power/3` to raise a number to some integral, non-negative power using only multiplication, e.g., `power(X, Y, Result)` should unify `Result` with X^Y.

Write a clause `fib/2` to compute Fibonacci numbers, e.g., `fib(X, N)` should unify `X` with the `N`th Fibonacci number.

Write a clause `prime/1` that tests whether a given integral, non-negative number is a prime number, e.g., the goal `prime(X)` is satisfied only if `X` is a prime number. A number `X` is a prime number if and only if there does not exist another number `Y`, $1 < Y < X$, such that `Y` divides `X` without remainder.

Part 4

This part is concerned with dates represented as the structure `date(Day, Month, Year)`, e.g., the due date of this assignment is represented by `date(16, 3, 2010)`.

Write a clause `valid_date/1` that is satisfied only if a given date on or after 1/1/1583 represents a day that existed under consideration of leap years, e.g. the query `valid_date (29, 2, 2010))` should not be satisfied and thus yield the answer `no`.

This requires an implementation of the Gregorian calendar [2]. The answer can be computed based on the following facts:

- 1/1/1583 was a Saturday.
- *"The Gregorian leap year rule is: Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100; the centurial years that are exactly divisible by 400 are still leap years. For example, the year 1900 is not a leap year; the year 2000 is a leap year."* [2]

Write a clause `weekday/2` that determines for any date on or after 1/1/1583 the weekday (`monday`, `tuesday`, …, `sunday`) of the date, e.g., `weekday(date(16, 3, 2010), X)` should bind `X` to `tuesday`.

Write a `clause last_workday_of_month/1` that determines whether a given date is the last workday (i.e., not a Sunday or Saturday; ignore government holidays) of a month.

Let a date's *sequence number* denote the number of days that have passed since 1/1/1583. **Write** a clause `prime_date/1` that is satisfied only if a given date's sequence number is prime.

Find you whether your birthday is a prime date (this may take a while, depending on your implementation of `prime/1`).

Part 5

Write a short essay (max. 1 page) on how Prolog could be embedded as part of a larger application to increase flexibility and simplify overall system complexity.

Be creative; surprise me with something that neither me nor any of the other students have thought of. However, your proposal should be conceivably implementable with algorithms known today (e.g., no super-human AI, no Skynet).

Extra Credit 1

Write a Prolog "library" (simply a file with a number of clauses) that offers an API to a **binary tree** abstract data type. You API should offer clauses for (at least) the following operations:

- `empty_tree(X)` unifies X with an empty tree.
- `singleton_tree(Elem, X)` unifies `X` with a tree that only contains `Elem`.
- `min_tree(Tree, X)` unifies `X` with the smallest element in `Tree`.
- `max_tree(Tree, X)` unifies `X` with the largest element in `Tree`.

- `add_tree(Elem, Tree, NewTree)` unifies `NewTree` with a tree that contains all elements of `Tree` and `Elem`.
- `print_tree(Tree)` outputs a Graphviz-compatible [3] description of `Tree`. The depiction should represent both the tree's structure and the value of each node.

Extra Credit 2

Extend your binary tree library to support **extraction**:

- `extract_min_tree(Tree, X, Rest)` unifies `X` with the smallest element in `Tree` and `Rest` with a tree that contains all elements in `Tree` with the exception of `X`.
- `extract_max_tree(Tree, X, Rest)` unifies `X` with the largest element in `Tree` and `Rest` with a tree that contains all elements in `Tree` with the exception of `X`.

Extra Credit 3

Modify your binary tree library to use a **balanced tree algorithm** of your choosing(e.g., red-black trees or AVL trees).

Guidelines

Your solution must be executable with SWI Prolog on the class host `stetson.cs.unc.edu`.

You cannot discuss Part 5 or any of the extra credit parts. You may discuss possible approaches to Parts 1-4 with other students, but you **cannot share source code**.

Telling someone what to type constitutes sharing of source code!

You may only use the standard library of Prolog. When solving the extra credit problems, the tree implementation should be done from "first principles" and not excessively rely on library data structures.

Hints:

- Skim Chapters 1, 2, and 3 in [1] before solving Parts 1 and 2.
- Skim Chapters 4 and 6 in [1] before solving Parts 3 and 4.
- Skim Chapter 5 before solving Extra Credit 1.
- Simplify complex clauses by composing them from simpler "helper clauses."

Deliverables

The Prolog source code for Parts 1–4 (and the extra credit problems, should you choose to solve them).

A text file (plain text of PDF) that contains a transcript of the queries of Part 1.

A text file (plain text of PDF) that contains a transcript of the queries of Part 2.

A text file (plain text or PDF) that contains the answers to Part 5.

Grading

Your solution will predominantly graded on correctness.

5 points — Part 1.	15 points — Extra Credit 1.
15 points — Part 2.	10 points — Extra Credit 2.
25 points — Part 3.	25 points — Extra Credit 3.
35 points — Part 4.	
20 points — Part 5.	

Style Guide

Please write your Prolog programs in a style such that your code resembles the examples in *Logic Programming with Prolog*.

Avoid cut (`!`) and negation (`\+`, `\=`) whenever possible.

References

1. *Logic Programming with Prolog*, Max Bramer, 2005. Springer Verlag, ISBN 1852339381. http://search.lib.unc.edu/search?R=UNCb5201841.
2. *Introduction to Calendars*, The United States Naval Meteorology and Oceanography Command, http://aa.usno.navy.mil/faq/docs/calendars.php.
3. *The DOT Language*, http://www.graphviz.org/doc/info/lang.html.