

Assignment 3

Due Thursday April 9 at 3:00pm.

No late assignments will be accepted.

No extensions can be given.

Programming questions must be answered in Prolog, with programs that show good declarative style. Marks will be deducted for programs that look like translated Java or C. Simple programs are preferred, especially small collections of simple rules. You should turn in three files: a text file of your solutions to the non-programming questions, a source listing of all your Prolog code, and a transcript of a terminal session with the Prolog interpreter. The terminal session should be short and should demonstrate that your programs and queries work correctly. Readability and well-written documentation are of great importance. *For full marks, all answers returned by Prolog must be correct; it is not sufficient for just the first answer to be correct.*

In this assignment, non-logical operations are rarely needed, and their use should be minimized. In particular, do *not* use `functor`, `arg`, `=..`, `assert`, `retract`, `fail` or `cut`. However, the built-in predicates `atom` and `number`, as well as negation and arithmetic operations, may be needed from time to time, and you may use them when necessary. Do not use the equality predicate, `=`, unless it reduces the size of your rules. Finally, do not use `;` (semicolon), which stands for logical “or” in Prolog.

The marker has a limited amount of time for each assignment, so it is your responsibility to provide documentation and testing that allows him to *quickly* evaluate your work. As with all work in this course, 20% of the grade is for quality of presentation.

1. Unification (8 points).

For each of the following pairs, state whether or not the atomic formulas unify. In each case, state what the most general unifier is, or show that unification is impossible. In this problem, U, V, X, Y, Z are variables, and a, b, c, d are constant symbols.

- (a) $q(a, X, b)$ and $q(Y, b, Z)$.
- (b) $q(U, b, f(c))$ and $p(a, Y, Z)$.
- (c) $q(a, X, b)$ and $q(Y, Y, Z)$.
- (d) $p(a, b, c)$ and $p(X, Y, X)$.
- (e) $p(a, Y, U, U, Y)$ and $q(X, X, Z, V, Z)$.
- (f) $p(a, Y, f(Y, c))$ and $p(X, b, Z)$.
- (g) $q(X, f(g(Z)), Z)$ and $q(Y, f(Y), h(a))$.
- (h) $q(g(a), f(X), X)$ and $q(Y, f(Y), h(a))$.

2. First-Order Logic (8 points).

Translate each of the following English statements into first-order logic. Do not use any function symbols, and use as few predicate symbols and constant symbols as possible. You may use the equality predicate if necessary.

- (a) Every person is male or female. (1 point)
- (b) Every person has a parent. (1 point)
- (c) A person is a mother if and only if she is female and has a child. (1 point)
- (d) A woman is married if and only if she has a husband, and a man is married if and only if he has a wife. (2 points)
- (e) Every person has exactly two parents. (3 points)

3. Databases (6 points).

Create a Prolog database of beers, bars, and drinkers. The database should define three predicates, as follows:

`visits(Drinker, Bar)` means that `Drinker` regularly visits `Bar`.

e.g., `visits(tony, red_lion)` means that Tony regularly visits the Red Lion.

`serves(Bar, Beer)` means that `Bar` serves `Beer`.

e.g., `serves(red_lion, molson)` means that the Red Lion serves Molson.

`likes(Drinker, Beer)` means that `Drinker` likes `Beer`.

e.g., `likes(tony, heineken)` means that Tony likes Heineken.

Your database should contain at least 30 atomic formulas using the three predicates above. Choose the data so that the queries in the questions below return interesting non-trivial answers (e.g., not always an empty answer, not always the same answer, not always a single answer).

4. Database Queries (5 points).

Without adding any rules to your Prolog database, pose queries to Prolog that retrieve the following information (and no more):

- (a) Those drinkers who like Bass.
- (b) Those beers that Tony likes.
- (c) Those bars that serve Michelob.
- (d) Those beers that both Tony and Albert like.
- (e) Those bars that Tony visits that serve Duval.

Note: your solutions must be independent of the data stored in the database, as described below. You may use `setof` to eliminate duplicate answers if you wish.

5. Deductive Databases (13 points total).

Now write Prolog rules to define the predicates below. Test your answers by posing queries to your Prolog database. Your rules should work even if the data in the database is changed. For example, if the Red Lion ceases to serve Duval, or if the Pig and Whistle starts to serve it, then your rules for the predicate `duvalier` should still work *without modification*; so they must not mention the name of any bar that serves Duval. This property of a rulebase is called *data independence*. To test your rules properly, you may need to change the data in the database. Be sure to clearly indicate what data you are testing your rules on. You may use `setof` to eliminate duplicate answers if you wish.

`fair(Beer)`, which means that some bar serves `Beer` and some drinker likes it (regardless of whether he visits the bar). (1 point)

`awful(Beer)`, which means that some bar serves `Beer` but *no* drinker likes it (including those drinkers who do not visit the bar). (1 point)

`beer(X)`, which means that someone likes beer `X` *or* some bar serves it. (1 point)

`drinker(X)`, which means that person `X` likes some beer *or* visits some bar. (1 point)

`poor(Beer)`, which means that some bar serves `Beer` but some drinker does *not* like it (regardless of whether he visits the bar). (1 point)

`great(Beer)`, which means that some bar serves `Beer` and every drinker likes it (including those drinkers who do not visit the bar). (2 points)

`duvalier(Drinker)`, which means that `Drinker` visits a bar that serves Duval. (1 point)

`normal(Drinker)`, which means that `Drinker` visits a bar that serves a beer he likes. (1 point)

`picky(Drinker)`, which means that `Drinker` visits *only* those bars that serve a beer he likes. *i.e.*, every bar the drinker visits serves a beer he likes. (4 points)

6. Recursion in Deductive Databases (15 points total).

The predicate `child(X,Y)` means that person `X` is a child of person `Y`. For instance, the formulas `child(tony,gord)` and `child(tony,rose)` mean that Tony is a child of Gord and also of Rose.

- (a) (2 points) Using the `child` predicate, write a database of atomic formulas that represents a small family tree spanning four generations of Roman nobility. The people at the bottom of the tree are thus the great-grand-children of the people at the top of the tree. Don't worry about historical accuracy, but to give the marker some common references, put the names Romulus and Remus in the first generation (the top of the tree); put the names Cicero and Scipio in the second generation; put the names Julius, Augustus, and Livia in the third generation; and put the names Caligula, Drusus, Nero, Germanicus, and Claudius in the fourth generation (the bottom of the tree). Fill out the family tree with at least ten other names. In addition, to represent inbreeding amongst the nobility, your family "tree" must be an acyclic graph. You should construct this graph so that the queries in the rest of this question test your program. The queries should return interesting and non-trivial answers.
- (b) (7 points total) Write a Prolog program that defines a predicate `descendant(X,Y)`, which means that person `X` is a descendant of person `Y` (3 points). This program should be data independent; that is, it should work for *any* database made from the `child` predicate, even if it represents a thousand generations. Using the `descendant` predicate, pose queries to your Roman database that retrieve the following data (1 point each):
- i. The descendants of Cicero.
 - ii. Those people who are descendants of both Romulus and Remus.
 - iii. The ancestors of Augustus.
 - iv. Those people who are ancestors of both Nero and Caligula.
- (c) (6 points total) The predicate `admires(X,Y)` means that person `X` admires person `Y`, and the predicate `disowned(X,Y)` means that parent `Y` has disowned his child, `X`. Add atomic formulas to your Roman database to show who admires who, and who has disowned who (2 points). Again, don't worry about historical accuracy, but choose your data so that the queries you ask return interesting and non-trivial answers. Finally, add three rules to your database, to represent the following knowledge:
- i. All the children of Augustus admire Julius. (1 point)
 - ii. Everyone admires Cicero, except for Caligula. (1 point)
 - iii. A person admires everyone that his parents admire, unless that parent has disowned him. (2 points)

Thus, some admiration facts are stored in the database, and others are derived by rules. Test your rules by posing Prolog queries.

7. Function Symbols and Lists (10 points total).

In Prolog, geometric figures can be represented as function terms. For example, `circle(4)` can represent a circle of radius 4, and `rectangle(5,8)` can represent a rectangle of length 5 and width 8.

- (a) (5 points) Write a set of Prolog rules that computes the area of geometric figures. That is, your program should define a predicate, `area(F,A)`, which means that `A`

is the area of figure F. Thus, `area(rectangle(4,6),24)` is true. Your program should be able to handle circles, squares, rectangles, parallelograms and at least one kind of triangle. Keep it simple, and don't use any fancy trigonometry. Assume that $\pi = 3.14$.

- (b) (5 points) Write a Prolog program that computes the total area of a list of geometric figures. That is, define a predicate `total(L,A)` where `L` is a list of figures, and `A` is the sum of their individual areas. Thus, the following query:

```
total([square(3),square(4),rectangle(2,4)],Area)
```

should return with `Area = 33`.

8. Structural Recursion and Mutual Recursion (16 points total)

Let `two` and `three` be function symbols of arity 2 and 3, respectively. We define a 2-3 tree to be a ground function term in which `two` and `three` are the only function symbols (other than constants). For example, `two(a,b)`, `three(a,b,c)`, `two(a,three(b,c,d))` and `three(two(a,b),three(a,b,c),e)` are all 2-3 trees. Intuitively, a 2-3 tree represents a tree (in the graph-theoretic sense) in which all internal nodes have 2 or 3 children. In particular, the function term `two(X,Y)` represents a node with two children (`X` and `Y`), and the function term `three(X,Y,Z)` represents a node with 3 children (`X`, `Y` and `Z`). Constant symbols represent the leaves of a 2-3 tree. In the following questions, you may assume that `X` and `Y` are 2-3 trees, so no error checking is required.

- (a) (8 points) Define a Prolog predicate `rev(X,Y)` that is true if and only if `Y` is the result of reversing the order of all the children in all the nodes of tree `X`. Here are some examples from a working program:

```
| ?- rev(two(a,b),Y).
    Y = two(b,a)
| ?- rev(three(a,b,c),Y).
    Y = three(c,b,a)
| ?- rev(two(a,three(a,b,c)),Y).
    Y = two(three(c,b,a),a)
| ?- rev(three(two(a,b),two(b,c),two(c,d)),Y)
    Y = three(two(d,c),two(c,b),two(b,a))
```

- (b) (8 points) Define a Prolog predicate `sub1(X,A,B,Y)` that is true iff `Y` is the result of replacing every occurrence of leaf `A` by `B` in tree `X`. Here are some examples from a working program:

```
| ?- sub1(three(a,a,a),a,b,Y).
    Y=three(b,b,b)
| ?- sub1(three(c,c,c),a,b,Y).
    Y=three(c,c,c).
| ?- sub1(two(a,three(a,b,c)),a,b,Y).
    Y = two(b,three(b,b,c))
```

Cover sheet for Assignment 3

Complete this page and hand it in during tutorial.

Name: _____
(Underline your last name)

Student number: _____

I declare that this assignment is solely my own work, and is in accordance with the University of Toronto Code of Behavior on Academic Matters.

Signature: _____