

## Exercise 2: ‘Introduction to Prolog – Part 2’

11 November 2013

**Notation** When specifying a relation/program to be defined by Prolog clauses in this and subsequent exercises we will use argument annotations to indicate the intended uses of the definition. We write  $+X$  to indicate that an argument  $X$  will be given,  $-X$  to indicate that the argument  $X$  will be an unbound variable, and  $?X$  to indicate that it can be given or be an unbound variable. This is the notation used in the Sicstus Prolog manual when describing uses of the primitive (‘built in’) predicates.

1. An acyclic directed graph is represented by the following facts:

```
arc(a, b).      arc(b, c).      arc(b, d).
arc(c, f).      arc(d, f).      arc(c, e).
arc(f, e).
```

- (a) Write two Prolog rules in order to define the predicate `path( ?X, ?Z )` which holds when the graph contains a uniformly-directed path from `X` to `Z`. For instance, in the given graph there is a path from `b` to `f`.

Your program should refer only to the `path` and `arc` predicates.

- (b) Try out your program for the queries (finding all solutions):

- i. `path(b, f).`
- ii. `path(b, Z).`
- iii. `path(X, d).`

- (c) Enter and run queries asking if

- i. there is a path from `a` to `f` that passes through `d`;
- ii. there is a path of length 3 ending at `f` (assume each arc has unit length);
- iii. there is a cycle in the graph.

2. The Peano numbers are `0`, `s(0)`, `s(s(0))`, ... etc. representing 0, 1, 2, ... etc.

- (a) Write a Prolog program for `plus( ?X, ?Y, ?Z )` which holds when `X`, `Y` and `Z` are Peano numbers satisfying  $X + Y = Z$ . Use no predicate other than `plus`.

- (b) Try out your program for the queries

- i. `plus(s(0), s(s(0)), Z).`
- ii. `plus(s(0), Y, s(s(s(s(0))))).`
- iii. `plus(X, Y, s(s(s(s(0))))).`

- (c) Write a single rule defining `odd( ?X )` in terms only of `plus`, where `odd( X )` expresses that `X` is an odd number. Test it with some queries.

3. Write a (recursive) program for `ones_zeros( ?X )` which holds when `X` is a list each of whose members, if any, is either 1 or 0.
4. Write a program for `hasdups( ?X )` which holds when list `X` contains at least two occurrences of some member. You can use `member/2`, which is a built-in predicate in Sicstus Prolog (but not necessarily in other Prolog implementations).
5. Write a recursive program for `prod(+L, ?P)` which takes a non-empty list `L` of numbers and returns the product `P` of all the numbers in the list. For example, the query `prod( [7, 2, -3], Prod )` returns `Prod = -42`. Refer only to `prod/2` and the built-in predicate `is/2`.
6. The predicate `contains( List1, List2, N )` holds when the list `List2` is a sublist of the list `List1`; the first element of `List2` occupies position `N` in `List1`.

```
contains( [a,b,c,a,d,e,f,g], [a,d,e,f,g], N )
    N = 4
contains( [a,b,c,a,d,c,a,b,c], [a,b], N )
    N = 1
    N = 7
contains( [a,b,c,e,d,f], [c,e,g], N )
    no
```

Define `contains( +List1, +List2, ?N )` in a single rule using `append/3` and `length/2`. Test it with various queries.

7. Write a program `add_poly(+P1,+P2,?P)` to perform polynomial addition, where all the coefficients and powers are integers e.g.

$$(3x^3 + 4x^2 + 5x + 6) + (7x^2 + 5x + 3) = (3x^3 + 11x^2 + 10x + 9)$$

One possible representation of the polynomial is as a list of tuples of the form `(C,I)` where `C` is the coefficient of the `I`th power of  $x$ , and where the pairs are ordered by decreasing powers of  $x$ . For example:

```
add_poly( [(3,3),(2,1),(6,0)], [(7,2),(8,1),(15,0)], P ).
    P = [(3,3),(7,2),(10,1),(21,0)]
```

You can assume that the tuples representing the two input polynomials are correctly ordered when `add_poly/3` is called.

*Optional* If you want to relax that assumption, you will have to order the tuples of the two input polynomials first. You can write your own ordering program, or use the built-in Sicstus predicate `sort/2`. (See manual). Try both.