Ben-Gurion University of the Negev
Department of Computer Science
Logic programming

Assignment 1, Semester B, 2023

Deadline: April 16 at 23:00

1 Before Solving

Some Prolog Notations

The exercise uses a few notations that you have seen in class. We repeat those notations here for convenience:

1. The notation `Pred/N` where `Pred` is a functor name and `N` is a natural number denotes a functor named `Pred` with arity `N`. For example, `append/3` denotes a functor named `append` with arity 3. We use this notation also for terms.

2. In addition, we use the mode notation, which denotes which arguments are instantiated during a predicate call. For example `append(+,+,-)` means that the predicate `append/3` may be called when the first two arguments are instantiated and the last argument may be uninstantiated. Notice that a predicate may support more than one mode. For example, `append/3` also supports mode `append(-,-,+)`.

Restrictions

There are a few restrictions you must follow in this assignment:

1. Make sure that your code is well documented and clear. Pay attention to style (we will when we check your code!). Credit will be given to elegant code writing.

2. **Do not use** constructs that we did not yet cover in class (no cuts, no findall, no if-then-else, no external libraries, and no negation of any kind – which includes the negation operators such as `\+` and the the inequality operators such as `\=`).

3. You may use the following external predicates: `append/3`, `reverse/2`, `member/2` and any other predicate which was taught in class. You may use `between/3` and Prolog arithmetics, including the arithmetic inequality operator `=\=`. **In tasks 1–4 you may not use any Prolog builtins or interpreted symbols**.

Unary Notation: the alphabet consists of `0/0` and `s/1` representing the zero constant and the successor function respectively. Examples are `0`, `s(0)`, `s(s(0))`, etc. The integer value associated with a term of this alphabet is the number of `s` symbols it contains. We have seen in class the predicates `nat/1`, `unary_plus/3`, `unary_times/3` and `unary_leq/2` which specify respectively the set of natural numbers and the relations defining addition, multiplication and less-equal of natural numbers.

Task 1: Unary Square Root (10%)

Given two natural numbers n and k, k is the integer square root of n if $k * k \leq n$ and $(k + 1) * (k + 1) > n$. Write a Prolog predicate `unary_sqrt(N,K)` with mode `unary_pow(+,-)` which succeeds if and only if `K` is the integer square-root of `N`, where `N` and `K`, are unary numbers. You may assume that `N` and `K` are legal representations of natural numbers in unary notation. Your implementation must produce legal representations as well.

```
?- unary_sqrt(s(s(s(s(0)))), K).
 K = s(s(0));
 false.
?- unary_sqrt(s(s(s(s(s(0))))),K).
K = s(s(0)) ;
false.
```

Task 2: Unary Divisor (10%)

For integers n and k, k is a divisor of n if there exists r such that $n = k \cdot r$. Write a Prolog predicate `unary_divisor(N,K)` with mode `unary_divisor(+,-)` which given an integer `N` in unary representation unifies `K` with every possible divisor of `N` upon backtracking. You may assume that `N` is in legal representation of natural numbers in unary notation. Your implementation must produce legal representations as well.

For example, (your solution might return these values in any order).

```
% divisors of 12: 1, 2, 3, 4, 6, 12
?- unary_divisor(s(s(s(s(s(s(s(s(s(s(s(s(0))))))))))), X).
X = s(0) ;
X = s(s(0)) ;
X = s(s(s(0))) ;
X = s(s(s(s(0)))) ;
X = s(s(s(s(s(0))))) ;
X = s(s(s(s(s(s(s(s(s(s(s(s(0)))))))))))) ;
false ;
```

Binary Notation: The alphabet consists of the symbols `[]/0`,`[|]/2`, 0/0 1/0, representing bit sequences as lists of zero and one bits (least significant bit first). The number 0 is represented as `[]`, and the number 1 as [1]. For any bit sequence N representing a **positive** number, the sequence $[0|N]$ represents $2 \times N$. For any bit sequence N representing a number,$[1|N]$ represents $2 \times N + 1$. The following are the first 7 binary numbers:

`[]`, `[1]`, `[0,1]`, `[1,1]`, `[0,0,1]`, `[1,0,1]`, `[0,1,1]`. Notice that this representation of binary numbers is least significant bit first (LSBF). Notice also that that binary numbers have no leading zeros.

Task 3: Binary Addition (15%)

Write a Prolog predicate `binary_plus(X,Y,Z)` with mode `binary_plus(+,+,-)` which succeeds if and only if `X`, `Y` and `Z` are natural numbers in binary notation such that `X+Y=Z`.

There are several ways to implement binary addition, which vary widely in efficiency. Your solution will be graded, in part, on the efficiency of your addition algorithm.

```
?- binary_plus([1,0,1], [1,1,1], C).
 C = [0, 0, 1, 1] ;
 false.
```

Task 4: Binary multiplication (10%)

Write a Prolog predicate `binary_times(X,Y,Z)` with mode `binary_times(+,+,-)` which succeeds if and only if `X` and `Y` and `Z` are natural numbers in binary notation such that `X*Y=Z`.

Notice: There are several ways to implement binary multiplication, which vary widely in efficiency. Your solution will be graded, in part, on the efficiency of your multiplication algorithm.

```
?- binary_times([0,1,1],[1,0,1],Z).
Z = [0, 1, 1, 1, 1] ;
false.
```

Task 5: Generate Binary Tree from Traversals (10%)

Write a Prolog predicate `gentree(Preorder,Postorder,Tree)` with mode `gentree(+,+,-)` which succeeds if and only if the binary tree `Tree` has preorder and postorder traversals that correspond to `Preorder` and `Postorder`. If there is more than one tree with the given preorder and postorder traversals then your program should return them all.

```
%% ?- gentree([1,2,3],[2,3,1],T).
%% T = tree(tree(nil, 2, nil), 1, tree(nil, 3, nil)) ;
%% false.
```

Task 6: Primality Testing (10%)

A prime number $n > 0$ is an integer that has no divisors greater than one and smaller than n. Write a Prolog predicate `is_prime(N)` with mode `is_prime(+)` which succeeds if and only if `N` is a prime number. You do not need to apply a probabilistic algorithm (but you may). Please provide an explanation in comments about your primality testing method. What is the largest prime number that your program can test (in reasonable time). For example,

```
?- is_prime(2).
true.
?- is_prime(22).
false.
?- is_prime(26233793190084349893096347).
true.
?- is_prime(28769375361317015373302983).
false.
```

Golomb Ruler: A Golomb ruler is a list `Xs` of increasing integers such that no two pairs of elements are the same distance apart. The length of `Xs` is called the order of the ruler. The smallest element is customarily put at 0.

Task 7: Test Golomb Ruler (10%)

Write a Prolog predicate `not_golomb(Xs)` with mode `not_golomb(+)` which succeeds if and only if the List `Xs` is **not** a Golomb ruler. You may assume that `Xs` is a list of positive integers. For example,

```
?- not_golomb([0, 1, 4, 9, 15, 22, 32, 34]).
false.
?- not_golomb([0, 1, 6, 10, 23, 26, 34, 41, 53, 55]).
false.
?- not_golomb([0, 1, 4, 9, 15, 34, 32, 22]).
true.
?- not_golomb([0, 1, 4, 9, 15, 22, 32, 35]).
true.
```

Task 8: Find a Golomb Ruler (10%)

Write a Prolog predicate `golomb(N,Max,Xs)` with mode `not_golomb(+,+,-)` which succeeds if and only if the List `Xs` is a Golomb ruler of length `N` in which all elements are less or equal to `Max`. For example,

```
?- golomb(10,55,Xs).
Xs = [0,1,6,10,23,26,34,41,53,55] ;
Xs = [0,2,14,21,29,32,45,49,54,55] ;
false.
?- golomb(10,54,Xs).
false.
```

Notice: Grading will take into account the longest rulers it can compute with the smallest values of `Max`. Grading will also take into account if your program can prove non-existence of certain rulers. For example as in the query `?- golomb(10,54,Xs)`.

Task 9: Evaluate Expression (without parentheses) (10%)

Write a Prolog predicate `evaluate(Expression,Value)` with mode `evaluate(+,-)` which succeeds if and only if the arithmetic expression `Expression` given as a list with numbers and operators `*, +` evaluates to `Value`. Assume that `Expression` represents a legal and not empty expression. For example,

```
?- evaluate([2,'+',3,'*',4,'*',2,'+',3],Value).
Value = 29 ;
false.
```

Task 10: Diagonal Ordered Magic Square (10%)

An order n Diagonal Ordered Magic Square is an $n \times n$ matrix that contains all of the numbers between 1 and n^2 such that every row, column and both diagonals have the same sum. In addition, both of the diagonals increase (from left to right).

Write a Prolog predicate `doms(Xs)` with mode `doms(+)` which succeeds if and only if the matrix `Xs` is a diagonal ordered magic square of order $n > 2$.

```
For example,
?- doms(
    [[1, 10, 7, 16],
     [15, 8, 9, 2],
     [14, 5, 12, 3],
     [4, 11, 6, 13]]

true.
```

2 Grading & Procedures

After Solving :

When grading your work, an emphasis will be given on code efficiency and readability. We appreciate effective code writing. The easier it is to read your code — the more we appreciate it! Even if you submit a partial answer. So please indent your code, add good comments.

Procedure

Submit a single file called `ex1.pl` with the assignment's solution. Please include a header with following statement:

> /**** I, Name (ID number) assert that the work I submitted is entirely my own.
> I have not received any part from any other student in the class (or other source),
> nor did I give parts of it for use to others. I have clearly marked in the comments
> of my program any code taken from an external source. *****/

Submission is solo, i.e., you may *not* work in pairs. If you take any parts of your solution from an external source you must acknowledge this source in the comments. Please note that we test your work using a Linux installed SWI-Prolog (as in the CS Labs) – so please make sure your assignment runs on such a configuration.