Introduction to Logic Programming

Sample Quiz A

Duration: 3 hours

There are 4 questions in this Quiz. You must answer all questions. Marking will take into account correctness, efficiency and clarity. Clarity means that your code is clear in style, starts at the top of a page, and is clear to read. Correctness means that your solution provides all correct solutions (once) as alternatives (unless indicated otherwise). Efficiency, means clever use of difference lists instead of traversing lists when not necessary (and many other things).

If you write **"don't know"** as an answer to a (part of a) question then you will get 20% of the points for that question. **Do Well!**

1. Nonogram Line **25 points**

Part A. 10 points You are to write a predicate `nonogram_verify(Ns,N,Xs)` which given a (possinly empty) list of positive integer values $\mathtt{Ns} = [\mathtt{n}_1, \mathtt{n}_2, \ldots, \mathtt{n}_\mathtt{k}]$, a positive integer `N`, and a list `Xs`, verifies that `Xs` is a list of length `N` and of the form $\mathtt{Xs} = 0^* 1^{\mathtt{n}_1} 0^+ 1^{\mathtt{n}_2} 0^+ \cdots 0^+ 1^{\mathtt{n}_\mathtt{k}} 0^*$.

Part B. 10 points You are to write a predicate `nonogram(Ns,N,Xs)` which given a (possinly empty) list of positive integer values $\mathtt{Ns} = [\mathtt{n}_1, \mathtt{n}_2, \ldots, \mathtt{n}_\mathtt{k}]$ and a positive integer `N` assigns `Xs` to a list of the form $\mathtt{Xs} = 0^* 1^{\mathtt{n}_1} 0^+ 1^{\mathtt{n}_2} 0^+ \cdots 0^+ 1^{\mathtt{n}_\mathtt{k}} 0^*$. For example,

```
?-nonogram([2,3],7,Xs).
Xs = [1, 1, 0, 1, 1, 1, 0] ;
Xs = [1, 1, 0, 0, 1, 1, 1] ;
Xs = [0, 1, 1, 0, 1, 1, 1] ;
false.

?-nonogram([2,3],5,Xs).
false.
```

```
?-nonogram([2,3],8,Xs).
Xs = [1, 1, 0, 1, 1, 1, 0, 0] ;
Xs = [1, 1, 0, 0, 1, 1, 1, 0] ;
Xs = [1, 1, 0, 0, 0, 1, 1, 1] ;
Xs = [0, 1, 1, 0, 1, 1, 1, 0] ;
Xs = [0, 1, 1, 0, 0, 1, 1, 1] ;
Xs = [0, 0, 1, 1, 0, 1, 1, 1] ;
false.
```

2. diff/2 (direct (unary) encoding) **25 points**

Part A. 5 points You are to write a predicate `direct(Xs,N,Cnf)` which given an integer value `N` creates a bit vector `Xs` of length `N` and a `Cnf` which specifies that `Xs` represents an integer value in the range $[0..N]$ in the direct encoding.

Part B. 15 points You are to write a predicate `diff(Xs,Ys,Cnf)` which given two lists (bit vectors) $Xs = [X_1, \ldots, X_n]$ and $Ys = [Y_1, \ldots, Y_n]$ (you may assume that they are of the same length) generates a CNF formula that specifies that Xs and Ys represent different numbers in the direct encoding. In this question, you assume that the bit vectors `Xs` and `Ys` represent numbers in direct encoding.

Part C. 5 points You are to write a predicate `allDiff(XXs,N,Cnf)` which given a list of Prolog variables `XXs` and an integer `N` binds `XXs` to a list of bit-vectors (each variable in the given list `XXs` to a bit vector) and generates a `Cnf` which is satisfiable exactly when the list of bit vectors `XXs` represent integer values in the direct encoding taking values in the range $[0..N]$ which are all different from each other.

3. Lexicographic Order Encoding **25 points**

Part A. 15 points Your task is to write the Prolog predicate `lexLT(Xs, Ys, Cnf)` which given two length $n \geq 0$ vectors (of Boolean variables) `Xs` and `Ys` creates a `Cnf` which is satisfied exactly when the corresponding instances of the vectors satisfy `Xs` < `Ys` (in the lexicographic order). Note: if you want to view the vectors `Xs` and `Ys` as binary numbers, then view them as "most significant bit first".

Part B. 10 points Your task is to write the Prolog predicate `matrixLexLt(Matrix, Cnf)` which given a matrix of length $n \geq 0$ (of Boolean variables) generates a `Cnf` which is satisfiable if and only if the rows of Matrix are lexicographically ordered (meaning that the first row is lexicographically smaller than the second, and so on..).

4. Sorting Networks **25 points**

A sorting network is an abstract mathematical model of a network of wires and comparator modules that is used to sort a sequence of numbers. Each comparator connects two wires and sorts the values by outputting the smaller value to one wire, and a larger value to the other. A network of wires and comparators that will correctly sort all possible inputs into ascending order is called a sorting network.

A network is represented as a list of terms of the form `comparator(A,B,C,D)` where A and B are the inputs to a comparator and C and D the outputs. We also maintain a list of input variables, and a list of output variables. For example, a sorting network for 8 inputs might look like this:

```
In  = [X1, X2, X3, X4, X5, X6, X7, X8]
Out = [Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8]

Cs  = [comparator(X1, X2, T1, T2),     comparator(X3, X4, T3, T4),
       comparator(T1, T4, T5, T6),     comparator(T2, T3, T7, T8),
```

```
comparator(T5, T7, T9, T10),    comparator(T6, T8, T11, T12),
comparator(X5, X6, T13, T14),   comparator(X7, X8, T15, T16),
comparator(T13, T16, T17, T18), comparator(T14, T15, T19, T20),
comparator(T17, T19, T21, T22), comparator(T18, T20, T23, T24),
comparator(T9, T24, T25, T26),  comparator(T10, T23, T27, T28),
comparator(T11, T22, T29, T30), comparator(T12, T21, T31, T32),
comparator(T25, T29, T33, T34), comparator(T27, T31, T35, T36),
comparator(T33, T35, Y1, Y2),   comparator(T34, T36, Y3, Y4),
comparator(T26, T30, T37, T38), comparator(T28, T32, T39, T40),
comparator(T37, T39, Y5, Y6),   comparator(T38, T40, Y7, Y8)]
```

Part A. 5 points You are to write a predicate `bit_vector(N, Vector)` with mode `bit_vector(+,-)` which accepts as a parameter an integer `N` and unifies `Vector` with a list of length `N` with zeros and ones. The predicate should unify `Vector` with all possible lists of zeros and ones upon backtracking.

For example,

```
?- bit_vector(3, Vector).
Vector = [0,0,0] ;
Vector = [0,0,1] ;
Vector = [0,1,0] ;
Vector = [0,1,1] ;
Vector = [1,0,0] ;
Vector = [1,0,1] ;
Vector = [1,1,0] ;
Vector = [1,1,1] ;
false.
```

Part B. 10 points Write a Prolog predicate `apply_network(Cs,In,Out)` which executes a given network of comparators `Cs` to map a given sequence of inputs `In` to the corresponding outputs `Out`. You may assume that `Cs` is a legal network (contains no cycles). For example,

```
?- sorting_network(4,Cs,In,Out), In = [4,3,2,1], apply_network(Cs,In,Out).
Out = [1, 2, 3, 4]
Cs  = ....
```

Part C. 10 points Write a Prolog predicate `is_a_sorting_network(Cs, In, Out)` which indicates if a given network of comparators `Cs`, is a sorting network. The predicate accepts `In` the variables which represent the network inputs, `Out` the variables which represent the network outputs, and `Cs` the list of the network comparators. The predicate fails if `Cs` is *not* a sorting network, otherwise if the network is a sorting network the predicate succeeds. This predicate must be deterministic. You may assume that `Cs` is a legal network representation.

Notice: Due to the zero-one theorem – it is enough that you test only sequences that are composed of 0s and 1s.