1 Natural Numbers (20 points)

The following predicate defines the natural numbers

\[
\text{nat}(0), \\
\text{nat}(s(X)) :- \text{nat}(X).
\]

1.1 Natural Number Average

You are to write a Prolog predicate \texttt{average(X,Y,Z)} which defines the relation: \(Z\) is the natural number average of \(X\) and \(Y\). A query to \texttt{average(X,Y,Z)} should terminate correctly when \(X\) and \(Y\) are given natural numbers. For example,

\[
\text{?- average}(s(s(0)),s(s(s(s(0)))),Z). \quad \text{%% average of 2,4 is 3} \\
Z = s(s(s(0)))
\]

\[
\text{average}(s(s(0))),s(s(s(s(0)))),Z). \quad \text{%% average of 3,4 is 3} \\
Z = s(s(s(0)))
\]

The only function symbols allowed in your solution are \texttt{s/1} and \texttt{0/0}. The only predicate symbol allowed is \texttt{average/3}. You may not use a cut (or an if-then-else). Your solution should not provide redundant answers.

1.2 Natural Number Division

The natural division of natural numbers \(m\) and \(n\) is the largest natural number \(d\) such that \(m \times d < n\). Note that \(n\) must be larger than 0. You are to write a Prolog predicate \texttt{divide(X,Y,Z)} which defines the relation: \(Z\) is the natural division of \(X\) and \(Y\). A query to \texttt{divide(X,Y,Z)} should terminate correctly when \(X\) and \(Y\) are given natural numbers. For example,

\[
\text{?- divide}(s(s(s(s(s(0))))),s(s(0)),Z). \\
Z = s(s(0))
\]

\[
\text{?- divide}(s(s(s(s(0)))),s(s(0)),Z). \\
Z = s(s(0))
\]

\[
\text{?- divide}(s(s(0)),s(s(s(s(0)))),Z). \\
Z = 0
\]

\[
\text{?- divide}(s(0),0,Z).
\]

No

The only function symbols allowed in your solution are \texttt{s/1} and \texttt{0/0}. You may not use a cut (or an if-then-else). Your solution should not provide redundant answers.

1.3 Powers of 2 (10 points)

You are to write a Prolog predicate \texttt{exp2(N,Y)} which given a natural number \(N\) instantiates \(Y\) to the value \(2^N\). For example,

\[
\text{?- exp2}(s(s(0))),Y). \\
Y = s(s(s(s(s(s(s(0)))))))
\]

\[
\text{?- exp2}(s(0)),Y). \\
Y = s(s(s(s(0))))
\]
1.4 Integer log (15 points)

The integer log base 2 of a natural number \(Y\) is the largest natural number \(N\) such that \(2^N \leq Y\). You are to write a Prolog predicate \(\text{log2}(Y,N)\) which given a natural number \(Y\) instantiates \(N\) to the natural log base 2 of \(Y\). For example,

?- \text{log2}(0,N).  
No  

?- \text{log2}(\text{s(s(s(s(s(0))))))),N).  
\(N = \text{s(s(0))}\)

?- \text{log2}(\text{s(s(0))},N).  
\(N = \text{s(0)}\)

?- \text{log2}(\text{s(s(0))),N).  
\(N = \text{s(0)}\)

?- \text{log2}(\text{s(s(s(0)))),N).  
\(N = \text{s(s(s(0)))))\)

?- \text{log2}(\text{s(s(s(s(s(s(s(0))))))))},N).  
\(N = \text{s(s(s(s(s(s(0)))))))}\)

?- \text{log2}(\text{s(s(s(s(s(s(0))))))),N).  
\(N = \text{s(s(s(s(s(0))))))}\)

2 Rotate (10 points)

We say that a list \(Y\) is a rotation of a list \(X\) if we can obtain \(Y\) from \(X\) by shifting the elements of \(X\) to the right moving the elements which fall of the end to front of the list. Write a Prolog predicate \(\text{rotate}(X,Y)\) which defines the rotations of a list \(X\). For example, of how the program should behave, consider the following query and its answers:

?- \text{rotate}([1,2,3],Y).  
\(Y = [1,2,3]\) ? ;  
\(Y = [2,3,1]\) ? ;  
\(Y = [3,1,2]\) ? ;  
no

?- \text{rotate}([],Y).  
\(Y = []\) ? ;  
no

The order in which you get the answers does not matter. Your program does not have to work backwards. But you do have to explain exactly how it behaves when it is run backwards.
3 Reverse the Evens

Write a Prolog predicate rev_even(A,B) that reverses the elements in the even positions of a list. For example, of how the program should behave, consider the following query:

```
| ?- rev_even([1,2,3,4,5,6],X).
X = [1,6,3,4,5,2] ? ;
```

no

For 80% of the credit write a straightforward solution which maybe does not work backwards. For 100% of the credit write a solution which runs in both directions and takes exactly N+1 unification operations. In other words you should traverse the input just once.

```
| ?- rev_even(X,[1,6,3,4,5,2]).
X = [1,2,3,4,5,6] ? ;
```

no

4 Powerset

The powerset of a set A is the set B of all subsets of A. Write a Prolog predicate pset(A,B) such that B is the powerset of A. For example, of how the program should behave, consider the following query:

```
| ?- pset([1,2,3],X).
X = [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]] ?
```

```
| ?- pset([],X).
X = [[]] ?
```

```
| ?- pset([1,2],X).
X = [[],[1],[2],[1,2]] ?
```

Note that the order of the elements in the powerset is not important.
5  Partition

A partitioning of a set A is a set of disjoint nonempty subsets of A which contain all of the elements of A. In this question we will represent sets of elements as a lists. You may assume that there are no repetitions of elements in the list representation.

Write a Prolog predicate partition(A,B) such that B is a partition of the set A. Make sure that your program will give all partitions of A as alternatives without repetitions. For example, of how the program should behave, consider the following query:

\[ ?- \text{partition([a,b,c],X).} \]
\[ X = \text{[a,b,c]} \] ? ;
\[ X = \text{[b,c],[a]} \] ? ;
\[ X = \text{[a,c],[b]} \] ? ;
\[ X = \text{[c],[a,b]} \] ? ;
\[ X = \text{[c],[b],[a]} \] ? ;
\[ \text{no} \]

6  K from N

Define the relation \( \text{subsets}(Xs,K,Ys) \) which, assuming that \( Xs \) contains a list of distinct symbols and \( K \) is a natural number in \( s \) notation, gives a list \( Ys \) of all subsets of \( Xs \) of length \( K \). The order in the elements of the lists does not matter. For example:

\[ ?- \text{subsets([a,b,c,d],s(0)),L).} \]
\[ L = \text{[b,a],[c,a],[d,a],[c,b],[d,b],[d,c]} \] ;
\[ \text{no} \]

- For (at most) 10 points you may solve this question using builtin Prolog constructs (such as \text{findall});
- For (at most) 15 points your solution should be in pure Prolog without using any builtin predicates. But you may define and use a relation that appends lists (in linear time);
- For (at most) 20 points your solution should be in pure Prolog without using any builtin predicates and it should append lists if at all using a constant time operation.
7 Compress

Consider a sequence of letters (a–z) such as xcaabaabaabcadacaabaabaabcada-
cady. The sequence could be compressed by a simple algorithm which replaces
n consecutive symbols by that symbol followed by n. For example ppppppp can
be replaced by p7. The same compression can be applied to subsequences. For
egative ababab can be replaced by (ab)3 (which contains 5 characters. So the
long sequence above can be compressed to x(c(a2b)3c2(ad)2)y, but this is not
the shortest compression because (ad)2 has one more character than adad.

You should write a predicate compress/2 which takes a sequence, given as a
list, and outputs an equivalent compressed list of shortest length.

?- compress([x,c,a,a,b,a,a,b,a,a,b,c,c,a,d,a,d,c,a,a,b,a,a,b,a,b,a,b,c,c,a,d,a,d,y], L).
L = [ x,(,c,(,a,a,b,),3,c,2,a,d,a,d,),2,y ]

There are other possible answers.

8 Word Matching

In this question you are requested to write a predicate match(Pattern, Words)
which given a list of Words (for example Words = [this, is, a, list, of, words])
and a list, Pattern, of words and variables (for example Pattern = [X, a, Y, of, Z])
- matches the variables to corresponding sublists of words. The specification of
the matcher is best described by a few examples:

?- match([X,a,Y,of,Z],[this,is,a,list,of,words]).
  X = [this,is],
  Y = [list],
  Z = [words] ? ;
  no

?- match([X,a,Y],[here,is,a,duck]).
  X = [here,is],
  Y = [duck] ? ;
  no

?- match([X,a,X],[a,joke,is,a,joke]).
  no

?- match([X,is,X],[a,joke,is,a,joke]).
  X = [a,joke] ? ;
  no

?- match([X,joke,Y],[a,joke,is,a,joke]).
You might want to make use of the predicate \texttt{nonvar(X)} which succeeds if \(X\) is bound to a nonvariable term and fails otherwise.

9 Merge Sort

1. Define a predicate \texttt{split/3}, such that \texttt{split(Xs,A,B)} is true only if the elements of the list \(Xs\) are distributed into the lists \(A\) and \(B\), under the constraint that the lengths of \(A\) and \(B\) differ by at most one. Note that this relation does not have to give all of the possible distributions.

2. Define a predicate \texttt{merge/3} as follows. Assuming that \(A\) and \(B\) are sorted lists of elements, \texttt{merge(A,B,C)} is true if and only if \(C\) is a sorted list containing all the elements of \(A\) and \(B\). For example,

\[
\texttt{merge([3,9,11],[2,7,10],[2,3,7,9,10,11])}
\]

3. Define a predicate \texttt{mergesort/2} which implements the mergesort algorithm. You may use the predicates \texttt{split/3} and \texttt{merge/3} from parts (a) and (b) respectively, even if you did not answer these parts.
10 Natural Numbers and Lists

The following predicate defines natural numbers and the lists of natural numbers

\[
\begin{align*}
\text{nat}(0). & \quad \text{nlist}([]). \\
\text{nat}(\text{s}(X)) :& - \text{nat}(X). & \quad \text{nlist}([X|Xs]) :& - \text{nat}(X), \text{nlist}(Xs).
\end{align*}
\]

Unordered pairs (10 points)

Let \( L \) be a given list of natural numbers. Write a predicate \text{unordered}(X,Y,L) which specifies that \( X \) is greater than \( Y \) and occurs before \( Y \) in list \( L \). For example:

\[\begin{align*}
X &= s(0), \ Y = 0; \\
X &= s(s(0)), \ Y = s(a(0)); \\
\text{No}
\end{align*}\]

\[\begin{align*}
X &= s(0), \ Y = 0; \\
X &= s(s(s(0))), \ Y = s(0); \\
X &= s(s(s(0))), \ Y = s(s(0)); \\
\text{No}
\end{align*}\]

Note: repetitions in your answers are allowed only if they are due to repetitions in the given list \( L \).

Unordered degree (10 points)

We define the unsorted degree of list of natural numbers \( List \) to be the number of pairs of elements \((X,Y)\) such that \text{unordered}(X,Y,List) holds. Write a predicate \text{degree}(List,N) which computes the unsorted degree of a list of natural numbers. For example:

\[\begin{align*}
X &= [s(0), 0, s(s(s(0))), s(s(0))]; \\
N &= 2; \\
\text{No}
\end{align*}\]

\[\begin{align*}
X &= [s(0), 0, s(s(s(0))), s(0), s(s(0))]; \\
N &= 3; \\
\text{No}
\end{align*}\]
11 Degree of Unsortedness

Define a predicate `unsort(Xs, D)` that, given a list of numbers `Xs`, finds its unsort degree `D`. The unsort degree of a list is the number of pairs of element positions in the list such that the first position precedes the second in the list, but the number occupying the first position is greater than the number occupying the second position. Some examples:

```
| ?- unsort([1,2,3],K). | ?- unsort([2,1,4,3],D). | ?- unsort([4,3,2,1],A).
K = 0 ? ;                    D = 2 ? ;                    A = 6 ? ;
no                       no                       no
```

12 Longest decreasing subsequence

Write a Prolog predicate `ld/2` which given a sequence of different numbers as first argument, determines a longest decreasing subsequence as second argument. Sequences are represented as lists, so a typical query might be:

```
?- ld([3,6,7,4,5,1,2],L).

With answer L = [7,5,2]. Also [6,4,1] is a correct answer (and some others). The input list is never empty.
```

13 Transpose

Write a Prolog predicate `transpose(Rows,Columns)` which given an $N \times M$ matrix `Rows` creates the corresponding transposed $M \times N$ matrix `Columns`. Your solution should work in both directions.

```
?- transpose([[a,b,c],[e,f,g]],M).
M = [[a, e], [b, f], [c, g]]

?- transpose(M,[[a, e], [b, f], [c, g]]).
M = [[a, b, c], [e, f, g]]
```
14 Chess Boards - Diagonals

You are to write a Prolog predicate `onDiagonal(X, Y, Matrix)` which given an \( n \times n \) board (Matrix) represented as a list of rows specifies that X and Y are on the same diagonal. Your program should return each pair (X,Y) only once (unless there is redundancy in the given Matrix). Do not return both (X,Y) and (Y,X). For example:

```prolog
?- onDiagonal(X, Y, [[a, b, c], [d, e, f], [g, h, i]]).
X = a, Y = e ; X = b, Y = d ; X = b, Y = f ;
X = a, Y = i ; X = c, Y = g ; X = d, Y = h ;
X = e, Y = i ; X = f, Y = h ; X = c, Y = e ;
X = e, Y = g ; No
```

15 Sudoku Boxes

A general “dimension \( n \)” Sudoku puzzle consists of an \( n^2 \times n^2 \) board and involves certain conditions on the: \( n^2 \) rows, \( n^2 \) columns and \( n^2 \) boxes, of the board.

You task is to write a Prolog predicate `box(N, Board, Box)` which given the dimension \( N \) and the \( N^2 \times N^2 \) Board provides the contents of a box of the board. If we ask for all of the alternative solutions we get the content of all of the boxes. For example,

```prolog
?- box(2, [[1, 2, 3, 4],
           [5, 6, 7, 8],
           [9, 10, 11, 12],
           [13,14,15,16]], Box).
Box = [1, 2, 5, 6] ;
Box = [3, 4, 7, 8] ;
Box = [9, 10, 13, 14] ;
Box = [11, 12, 15, 16] ;
No
```

Note that this predicate must work also for the case that the board is filled with variables.

```prolog
?- box(2, [[X1, X2, X3, X4],
           [X5, X6, X7, X8],
           [X9, X10, X11, X12],
           [X13, X14, X15, X16]], Box).
Box = [X1, X2, X5, X6] ;
Box = [X3, X4, X7, X8] ;
Box = [X9, X10, X13, X14] ;
Box = [X11, X12, X15, X16] ;
No
```
The \textit{direct-product} of two lists \(L_1\) and \(L_2\) of the same length is defined as a list of pairs, where the \(i\)-th pair contains the corresponding \(i\)-th elements of \(L_1\) and \(L_2\). e.g, the direct product of the lists \([1,2,3]\) and \([4,5,6]\) is \([(1,4),(2,5),(3,6)]\).

The \textit{reverse-direct-product} of the lists \(L_1\) and \(L_2\) (\textit{with equal or different lengths}) is defined as follows:

1. if the lists are not of the same length then extend the shortest list (by adding zeros at the end) so that the two lists will have the same length.
2. reverse the second list (\(L_2\)); and
3. apply the direct product on the first list (maybe extended) and the result of (2).

For example, the reverse-direct-product of the lists \([1,2,3]\) and \([4,5,6]\) is \([(1,6),(2,5),(3,4)]\), and of the list \([1,2,3]\) and \([5,6]\) is \([(1,0),(2,6),(3,5)]\).

Choose \textit{only one} of the following tasks:

\textbf{Task 1 (5 Points)}: Write a program \texttt{dp(L1,L2,L3)} which implements the relation “\(L_3\) is the direct-product of \(L_1\) and \(L_2\)”.

\textbf{Task 2 (15 Points)}: Write a program \texttt{rdp(L1,L2,L3)} which implements the relation “\(L_3\) is the reverse-direct-product of \(L_1\) and \(L_2\)”.

\textbf{Task 3 (25 Points)}: Write a program \texttt{rdp(L1,L2,L3)} which implements the relation reverse-direct-product, and respects the following:

- The complexity of the program (the number of derivations) should be \(N + c\) where \(N\) is the length of the longest list and \(c\) is a constant.
- Your program should work in all modes.

\textbf{Examples}:

\begin{verbatim}
| ?- rdp([1,2,3], [4,5,6], L3).
L3 = [(1,6), (2,5), (3,4)] ;
no
| ?- rdp([1], [2,3,4], L3).
L3 = [(1,4), (0,3), (0,2)] ;
no
| ?- rdp([1], [4,5,6], L3).
L3 = [(0,6), (0,5), (0,4)] ;
no
| ?- rdp([1], [], L3).
L3 = [(1,0)] ;
no
\end{verbatim}
| ?- rdp(L1,L2,[(2,0),(0,1)]).
L1 = [2],
L2 = [1,0] ? ;
L1 = [2,0],
L1 = [2,0],
L2 = [1,0] ? ;
no
| ?- rdp(L1,L2,[(2,0),(3,0),(4,1)]).
L1 = [2,3,4],
L1 = [2,3,4],
L2 = [1,0] ? ;
L1 = [2,3,4],
L2 = [1,0,0] ? ;
no

17 Cut

Consider the following program:

\[
\begin{align*}
top(X,Y) & :- p(X,Y). \\
top(X,X) & :- s(X). \\
p(X,Y) & :- true(1), q(X), true(2), r(Y). \\
p(X,Y) & :- s(X), r(Y). \\
q(a). \\
q(b). \\
r(c). \\
r(d). \\
s(e). \\
true(_). 
\end{align*}
\]

1. List (in the proper order) all of the answers to the query \(?- p(X,Y).\)

2. List (in the proper order) all of the answers to the query \(?- p(X,Y)\) when true(1) is replaced by cut;

3. List (in the proper order) all of the answers to the query \(?- p(X,Y)\) when true(2) is replaced by cut; and

4. List (in the proper order) all of the answers to the query \(?- p(X,Y)\) when both true(1) and true(2) are replaced by cut.

5. List (in the proper order) all of the answers to the query \(?- top(X,Y)\) when true(1) is replaced by cut.

6. List (in the proper order) all of the answers to the query \(?- top(X,Y)\) when true(2) is replaced by cut.
18 Triplet

Write a Prolog program which generates all triples \((X, Y, Z)\) which satisfy the conditions that

1. \(X, Y\) and \(Z\) are different digits (from \(0, \ldots, 9\)); and
2. \((10 \times X + Y) / (10 \times Y + Z) = X/Z\).

Your program should be queried as \(?-\) triplets and print on the screen the solutions row by row:

1 6 4
1 9 5 etc.
19 Gray Codes (generation)

Your first task is to count. Write a Prolog program defining the predicate `count(N, Numbers)` which given an integer $N > 0$ computes the list of $N$-digit binary numbers from 0 to $2^N - 1$ (in the order as you would count). To keep things simple we will represent an $N$-digit binary number as a list of $N$ binary digits. For example,

?- count(3, Ns).
Ns = [[0,0,0], [0,0,1], [0,1,0], [0,1,1], [1,0,0], [1,0,1], [1,1,0], [1,1,1]]

Your second task concerns Gray codes. An $N$-bit Gray code is a permutation of the list of $N$-bit binary numbers such that each two consecutive numbers differ in at most one position. The Gray code "wraps around:" When rolling over from the last number to the first, only one digit changes. Here is one example 3-bit Gray code.

[[0,0,0], [0,0,1], [0,1,1], [0,1,0], [1,1,0], [1,1,1], [1,0,1], [1,0,0]]

Another Gray code can be obtained by flipping 0’s to 1’s and 1’s to 0’s. There are several algorithms to create $n$-bit Gray codes. Lets think recursively,...: Say we have an $n$-bit Gray code. For example with $n = 2$ we have the list $L = [[0,0], [0,1], [1,1], [1,0]]$. Then the $(n+1)$-bit Gray code can be constructed as follows: Take two copies of $L$. In the first, attach to each binary number (list) a zero $L_1 = [[0,0,0], [0,0,1], [0,1,1], [0,1,0]]$ and in the second, attach to each number a one: $L_2 = [[1,0,0], [1,0,1], [1,1,1], [1,1,0]]$. There is a way to combine $L_1$ and $L_2$ to get the desired $n+1$-bit code.

Your task is to write the Prolog predicate `gray(N, List)` which receives an integer $N > 0$ and computes an $n$-bit Gray code in `List` using the above algorithm. Example:

?- gray(3, L).
L = [[0,0,0], [0,0,1], [0,1,1], [0,1,0], [1,1,0], [1,1,1], [1,0,1], [1,0,0]]

20 Gray Code (check)

Write a Prolog program which checks if a list of binary numbers (represented as lists of binary digits) represents a legitimate Gray code as defined in question 1. For example:

?- check([[0,1,1], [0,1,0], [0,0,0], [0,0,1], [1,0,1], [1,0,0], [1,1,0], [1,1,1]]).
yes
?- check([[0,0,0], [0,0,1], [0,1,1], [0,1,0], [1,1,0], [1,1,1], [1,0,1], [1,0,0]]).
yes
?- check([[0,0,0], [0,0,1], [0,1,0], [0,1,1], [1,0,0], [1,0,1], [1,1,0], [1,1,1]]).
no
?- check([[0,0,0], [0,0,1], [0,1,1], [0,1,0]]) no
21 Gray Codes (generation)

An $n$-bit Gray code is a permutation of the list of $n$-bit binary numbers such that each two consecutive numbers differ in at most one position. The Gray code "wraps around." When rolling over from the last number to the first, only one digit changes. In this question we will represent an $n$-bit binary number as a list of zeros and ones. Here is one example 3-bit Gray code.

$$[[0,0,0],[0,0,1],[0,1,1],[0,1,0],[1,1,0],[1,1,1],[1,0,1],[1,0,0]]$$

There are several algorithms to create $n$-bit Gray codes. In this exam we will implement one particular algorithm to create one particular type of Gray code.

The idea is to first implement a predicate gray/2 such that gray(I,G) is in the relation if $G$ is the $(I+1)$-th element in the particular type of Gray sequence illustrated above. Here both $I+1$ and $G$ are represented as $n$-bit binary numbers and $I$ ranges from 0 to $2^n - 1$. For example, the following queries give the 1st and 5th elements in the 3-bit Gray code shown above:

?- gray([0,0,0],G1). %I=0
G1 = [0,0,0]

?- gray([1,0,0],G5). %I=4
G5 = [1,1,0]

To solve this task you should implement two operations on $n$-bit binary numbers:

1. $n$-digit integer division by 2: implement a predicate half(I,J) specifying that $n$-digit binary $J$ is half of $n$-digit binary $I$. Examples:

?- half([0,0,1,1],M).
M = [0,0,0,1]

?- half([0,0,1,0],M).
M = [0,0,0,1]

2. XOR: the xor of two bits is defined to be 1 if exactly one of the bits is 1 and otherwise it is defined to be 0. The xor of two binary numbers is defined “bitwise”. Implement a predicate xor(I,J,X) specifying that $X$ is the xor of the $n$-digit binary numbers $I$ and $J$. Example:

?- xor([0,0,1,1],[0,0,0,1],X).
X = [0,0,1,0]

Now we are ready to implement the predicate gray(I,G). The $n$-bit binary number $G$ corresponding to $I$ is obtained as the xor of $I$ with the integer division by 2 of $I$. This predicate does not have to work in both directions but you must specify what happens if we try to apply it backwards. For example what is the behaviour of

?- gray(1,[0,1,1]).

Implement a predicate graycode(N,List) such that List is the $N$-bit Gray code obtained by the above algorithm. For example:

?- graycode(3,L).
L = [[0,0,0],[0,0,1],[0,1,1],[0,1,0],[1,1,0],[1,1,1],[1,0,1],[1,0,0]]

You may use the predicates given in question 2.
22 Gray Codes (check)

To determine if a given list \( X = [b_1, \ldots, b_k] \) of \( n \)-digit binary numbers is a legitimate Gray code (not necessarily the same as the one shown above) one must check that the list satisfies several properties. A student who took moed aleph made the claim that if the list satisfies the following properties then it must be a Gray code:

1. the number of elements \( k \) in the list is equal to \( 2^n \) where \( n \) is the number of digits in each of the binary numbers in the given list.
2. all of the elements in the list are different.
3. each two consecutive elements \( b_i \) and \( b_{i+1} \) for \( 1 \leq i < k \) differ in exactly one position.

In particular, notice that the student claims that it is not necessary to check if the first and last elements differ in exactly one position. The student claims that because of the other conditions it is not required to check this.

Your task is to write a Prolog program to check if this is true. The predicate \( \text{test}(N, \text{BadCode}) \) should return a list of \( N \)-digit binary numbers which show that the student is wrong. For example,

\[
?- \text{test}(3,BC).
\]
\[
\text{BC} = [[0,0,0],[0,0,1],[0,1,1],[0,1,0],[1,1,0],[1,0,0],[1,0,1],[1,1,1]]
\]

You may use the following predicates if you like:

1. % two lists of digits differ in exactly one position
   \[
   \text{gray_pair}([X|Xs],[X|Ys]) :- \text{gray_pair}(Xs,Ys).
   \]
   \[
   \text{gray_pair}([X|Xs],[Y|Xs]) :- X \neq Y.
   \]
2. % Count is the ordered list of \( N \) digit binary numbers from % \( [0,0,...,0] \) to \( [1,1,...,1] \)
   \[
   \text{count}(N, \text{Count}) :- \text{findall}((\text{Digits}), \text{number}(N, \text{Digits}), \text{Count}).
   \]
   \[
   \text{number}(N, \text{Digits}) :- \text{Digits} \text{ is an } N\text{-digit binary number}
   \]
   \[
   \text{represented as a list binary digits}
   \]
   \[
   \text{number}(N, \text{Digits}) :- \text{length}(\text{Digits},N), \text{digits}(\text{Digits}).
   \]
   \[
   \text{digits}([]).
   \]
   \[
   \text{digits}([X|Xs]) :- \text{digit}(X), \text{digits}(Xs).
   \]
   \[
   \text{digit}(0).
   \]
   \[
   \text{digit}(1).
   \]
23 Number of BST’s

Write a prolog predicate \( bst(N, M) \) to compute the number \( M \) of binary search trees with \( N \) distinct (integer) nodes.

\[
?- bst(3, M).
\]
\( M=5 \)

24 Same Frontier 20 points

You should write a Prolog predicate \( sf/2 \) which receives as inputs two binary trees and indicates if the two trees have the same frontiers.

Assume that a binary tree is represented as in the following definition:

\[
tree(void).
\]
\[
tree( tree(X,L,R) ) :-
  tree(L), tree(R).
\]

For example,

\[
?- T1=tree(1,tree(2,void,void),tree(3,void,tree(4,void,void))),
  T2=tree(3,tree(2,void,void),tree(4,void,void)),
  sf(T1,T2).
\]
\( yes \)

\[
?- T1=tree(1,tree(2,void,void),tree(3,void,tree(4,void,void))),
  T3=tree(2,tree(3,void,void),tree(4,void,void)),
  sf(T1,T3).
\]
\( no \)

For 80% of the credit write the straightforward solution which computes the two frontiers and compares them. For 100% of the credit write a solution that stops traversing the trees as soon as it encounters an inconsistency between the two frontiers.

25 Tree by Level

Write a Prolog predicate \( bylevel(Tree, List) \) which traverses a tree level by level and returns the labels on the nodes as a list of lists of elements one list for each level. For example, if the predicate \( bst3(T) \) returns as alternatives the 5 binary search trees with 3 nodes shown above then we get the following behaviour for the predicate you are to write:

\[
| ?- bst3(T), bylevel(T,L).
L = [[1],[2],[3]],
T = tree(nil,1,tree(nil,2,tree(nil,3,nil))) ? ;
L = [[1],[3],[2]],
T = tree(nil,1,tree(tree(nil,2,nil),3,nil)) ? ;
\]
L = [[[2],[1,3]],
T = tree(tree(nil,1,nil),2,tree(nil,3,nil)) ;
L = [[[3],[2],[1]],
T = tree(tree(tree(nil,1,nil),2,nil),3,nil) ;
L = [[[3],[1],[2]],
T = tree(tree(nil,1,tree(nil,2,nil)),3,nil) ;
no

26 Trees as Graphs
Write a Prolog predicate treeEdges(T,Edges) which given a binary tree T computes its set of graph edges. For example
?- tree2edges(tree(c,void,tree(d,void,void)),Edges).
Edges = [edge(c, d)]
?- tree2edges(tree(d,void,void),Edges).
Edges = []
?- tree2edges(tree(c,tree(e,void,void),tree(d,void,void)),Edges).
Edges = [edge(c, d), edge(c, e)]

In this question graphs are not directed. An edge edge(a,b) indicates an edge between nodes a and b. Just a reminder: A binary tree is either an empty tree void or else of the form tree(Label,Left,Right) where Left and Right are binary trees. In this question you should assume that all of the nodes in the tree have different labels.

27 Centered Trees
The distance between two nodes u and v in a graph is the length of the shortest path between u and v (the length of a path is the number of edges in the path). The eccentricity of a node v is the maximal distance between v and any other node in the graph. A node with minimal eccentricity is named a center of the graph. There is a special class of trees, called centered trees which have exactly one center.

Your task is to write a Prolog predicate centered(T,Center) which when given a binary tree T determines if it is centered and gives the Center. You may use the predicate specified in the previous question. You should assume that all of the nodes in the tree have different labels and that there is at least one edge.

?- Tree = tree(a,tree(b,void,void),tree(c,void,void)), centered(Tree,C).
C = a
?- Tree = tree(a,tree(b,tree(c,void,tree(d,void,void)),
tree(e,void,void)),tree(f,void,void)),centered(Tree,C).
C = b
?- Tree = tree(a,tree(b,void,void),tree(c,void,tree(d,void,void)))
, centered(Tree,C).
No
28 Binary Trees

The following predicate defines the binary trees

```prolog
binTree(nil).
binTree(tree(Left,X,Right)) :-
    binTree(Left),
    binTree(Right).
```

28.1 Binary Search Tree (15 points)
Recall that a binary search tree is a binary tree such that each node \( N \) is a number larger than all nodes to its left, and smaller than all nodes to its right. Write a predicate `isBST(Tree)` which defines the set of binary search trees. For example

?- isBST(tree(tree(nil, 0, nil), 1, tree(nil, 2, nil))).

Yes

?- isBST(tree(tree(nil, 0, nil), 2, tree(nil, 1, nil))).

No

28.2 Perfect Binary Tree (15 points)
Recall that a perfect binary tree if all nodes in \( T \) have two children and all leaves in \( T \) are at the same depth. Write a predicate `perfect(Tree)` which defines the set of perfect binary trees. For example

?- perfect(tree(tree(tree(nil, f, nil), c, tree(nil, d, nil)), a, tree(tree(nil, e, nil), b, tree(nil, e, nil))).

Yes

?- perfect(tree(tree(tree(nil, f, nil), c, nil), a, tree(tree(nil, d, nil), b, tree(nil, e, nil))).

No

28.3 Tree Edges
An edge in a binary tree is a pair of nodes `edge(A,B)` such that \( A \) is the parent of \( B \). You are to write a Prolog predicate `edges(Tree,List)` which given a binary tree `Tree` instantiates `List` to the list of its edges (the order does not matter. For example

?- Tree = tree(  
    tree(  
        tree(nil, c, tree(nil, d, nil)),  
        b,  
        tree(nil, e, nil)  
    ),  
    a,  
    tree(nil, f, nil)  
  ),  
  tree2edges(Tree,List).

L = [edge(c, d), edge(b, e), edge(b, c), edge(a, f), edge(a, b)]
28.4 Binary Tree Traversal - BFS

You are to write a Prolog predicate `bft(Tree,List)` which given a binary tree computes its breadth first traversal as the list `List`. For example given:

```prolog
T1 = tree(tree(nil,a,nil), b, tree(nil,c,nil) ).
T2 = tree(nil,d,nil).
T3 = tree(T1,e,T2).
```

We expect:

```prolog
?- bft(T1,List).  % List = [b, a, c]
?- bft(T2,List).  % List = [d]
?- bft(T3,List).  % List = [e, b, d, a, c]
```

28.5 Binary Tree Traversal - pip-order

Consider a traversal for binary trees where for a tree of the form `tree(L,N,R)` we first visit the node `N`, then recursively visit the left subtree `L`, then visit again `N`, then recursively visit the right subtree `R` and finally visit again `N`. We call such a traversal a pip-order (pre-in-post order) traversal.

Write a Prolog predicate `pip(T,List)` which evaluates the pip-order traversal of a given binary tree, `T`. For full credit (25 points):

1. Your solution will use difference lists to implement a linear time algorithm.
2. Your program must work also in the reverse direction to build a binary tree given its pip-traversal (see example).

For example:

```prolog
?- pip(tree(tree(nil, d, nil), b, tree(nil, e, nil)),List).
List = [b, d, d, d, b, e, e, e, b];
No
?- pip(tree(tree(tree(nil, f, nil), c, nil), a, tree(nil, d, nil), b, tree(nil, e, nil)), List).
List=[a, c, f, f, f, c, c, a, b, d, d, b, e, e, e, b, a];
No
```

A correct solution which uses `append/3`, or does not work in the reverse direction will contribute 10 points. You may write two solutions to this question: One using `append`, and the other using difference lists. If only the first is correct then you will get 10 points.

29 Loops

You get a directed graph in the form (these are facts in the program!).
edge(a,b).
edge(b,c).
edge(c,c).
edge(a,d).
edge(d,a).

You are to write a predicate loops/1 that returns a possibly empty list of all minimal cycles in the graph. A cycle is a path which starts and ends with the same node. A minimal cycle does not properly contain any other cycle. Every minimal cycle should be given exactly once. But the order of the cycles and the order of the nodes in the cycle does not matter. For the above graph your program might return

$$\text{[[c,c], [a,d,a]] or [[c,c], [d,a,d]]}$$

or other variations.

### 30 Strongly Connected

Write a Prolog predicate scc/1 which given the specification of a directed graph $G$ determines if $G$ is strongly connected. Remember that a graph is strongly connected if there is a path between every pair of nodes. For example the queries:

?- scc([edge(a,b), edge(b,c), edge(c,b), edge(b,a), edge(d,a)]).
Yes

?- scc([edge(a,b), edge(b,c), edge(c,b), edge(b,a), edge(d,a)]).
No

### 31 Spanning spider

A spanning spider of a graph is a spanning tree (a subgraph that is a tree and contains all vertices) that is also a spider. A spider is a graph with at most one vertex whose degree is three or more. Not every graph has a spanning spider and you will write a predicate spanspid/0 which succeeds once if a given graph is a spanning spider and fails otherwise. The graph is given as a predicate consisting of facts edge/2. The edges are undirected. Three examples edge/2 sets with the query and answer are shown below:

1. edge(d,a). edge(d,b). edge(d,c). edge(d,e).
   edge(a,x). edge(b,y). edge(c,z).
   ?- spanspid.
   Yes

2. edge(a,e). edge(b,e). edge(e,f).
   edge(f,d). edge(f,c). edge(x,y).
   ?- spanspid.
   No

3. edge(a,b). edge(b,c). edge(a,c).
   edge(d,a). edge(d,b). edge(d,c).
   ?- spanspid.
   Yes
32 SAT Solving

Write a Prolog predicate sat/1 which given a formula \( F \) in conjunctive normal form checks if it is satisfiable.

A formula in conjunctive normal form is represented as a list of list of literals (a literal is a variable or its negation). The elements of the outer list are viewed as a conjunction. Each inner list is a disjunction. For example the CNF formula \((A + (\neg B)) \land ((\neg A) + B)\) is represented as \([[[A, \neg B], [\neg A, B]]]\).

A call to sat(F) should provide -1,1 bindings to the variables in F. For example:

```prolog
?- sat([[A,-B],[-A,B]]).
A = 1
B = 1 ;
A = -1
B = -1 ;
No
```

Your solution does not have to be efficient (we have sat solvers for that). It should be correct.

33 SAT solving

33.1 CNF (10 points)

Recall that: A literal is a propositional variable \( X \) or its negation \( \neg X \). A clause is a disjunction of literals and a conjunctive normal form (CNF) is a conjunction of clauses. We represent clauses as lists of literals and CNF’s as lists of clauses.

The following code was suggested to be a SAT solver that receives a propositional formula and assigns values -1,1 to its variables such that the formula is satisfied.

```prolog
sat_cnf([]).
sat_cnf([C|Cs]) :- sat_clause(C), sat_cnf(Cs).

sat_clause(C) :- member(1,C).
sat_clause(C) :- member(-(1),C).
```

If it works correctly then given the query

```prolog
?- sat_cnf([[X,Y,Z],[-X,-Y],[-X,-Z],[-Y,-Z]]).
```
Prolog might respond with

```
X = -1, Y = 1, Z = -1
```

Specify the precise result of the following two queries:

1. ```prolog
   ?- sat_cnf([[X,Y],[-X,-Y]]),
      writeln([X,Y]),
      fail.
```
2. ```prolog
   ?- sat_cnf([[X,Y,Z],[-X,-Y],[-X,-Z],[-Y,-Z]]),
      writeln([X,Y,Z]),
      fail.
```
33.2 DNF (10 points)

Recall that a literal is a propositional variable \(X\) or its negation \(-X\). Define a *term* to be a conjunction of literals and a disjunctive normal form (DNF) to be a disjunction of terms. In this question we represent terms as lists of literals and DNF’s as lists of terms. For example the list of lists \([[X,-Y],[\neg X,Y]]\) represents the formula \((X \land \neg Y) \lor (\neg X \land Y)\).

Your task is to write a simple SAT solver \texttt{sat\_dnf(DNF)} that receives as input a DNF and assigns values -1,1 to its variables such that the formula is satisfied. Your program should perform its task in linear time (in the size of its input). For example,

\[
\texttt{?- sat\_dnf([[X,-Y],[\neg X,Y]].} \\
X = 1, Y = -1 ; \\
X = -1, Y = 1 ; \\
\texttt{No}
\]

34 SAT solving

Recall that: A literal is a propositional variable \(X\) or its negation \(-X\). A clause is a disjunction of literals and a conjunctive normal form (CNF) is a conjunction of clauses. We represent clauses as lists of literals and CNF’s as lists of clauses.

34.1 Propagate (20 points)

You are to write the Prolog predicate \texttt{propagate(Lit,Cnf1,Cnf2)} which simplifies the conjunctive normal form \(CNF1\) given that the literal \(Lit\) is known to be true. The result of this simplification is \(Cnf2\). In case the result of propagation is \texttt{true} or \texttt{false} then \(Cnf2=[]\) and \(Cnf2=[[[]]\) respectively. For example (the order of the clauses is not important),

\[
\texttt{?- propagate(X,[[\neg X,Y],[\neg X,Z],[Y,Z]],C).} \\
C = [[Y],[Z],[Y,Z]] \\
\texttt{?- propagate(-X,[[\neg X,Y],[\neg X],[Y,Z]],C).} \\
C = [[Y,Z]] \\
\texttt{propagate(X,[[X,Y],[X,Z]],C).} \\
C = [] \\
\texttt{?- propagate(X,[[\neg X,Y],[\neg X]],C).} \\
C = [[[]]]
\]

34.2 To DIMACS

In DIMACS format a CNF is represented by a sequence of integers. Literals are represented by integers. The positive integer \(n\) represents the propositional variable \(x_n\), and the negative integer \(-m\) represents \(-x_m\). Clauses are represented by a sequence of nonzero integers. In DIMACS format the sequences representing different clauses are seperated by 0’s. So for example the sequence

\[
1\ 2\ 3\ 4\ -1\ -2\ 0\ -1\ 3\ 0\ -2\ -3\ 0
\]
represents the cnf \([\{X_1,X_2,X_3\},\{\neg X_1,\neg X_2\},\{\neg X_1,X_3\},\{\neg X_2,\neg X_3\}\] in our representation.

You are to write a Prolog predicate `dimacs(Cnf,List)` which given a conjunctive normal form `Cnf` in our representation creates a list of integer values of the DIMACS representation. For example,

```prolog
?- dimacs([[X,Y],[-X,Y,Z]],D).
D = [1, 2, 0, -1, 2, 3, 0]
```

### 34.3 From DIMACS

Recall that: A literal is a propositional variable \(X\) or its negation \(\neg X\). A clause is a disjunction of literals and a conjunctive normal form (CNF) is a conjunction of clauses. We represent clauses as lists of literals and CNF’s as lists of clauses.

In DIMACS format a CNF is represented by a sequence of integers. Literals are represented by integers. The positive integer \(n\) represents the propositional variable \(x_n\), and the negative integer \(-m\) represents \(\neg x_m\). Clauses are represented by a sequence of nonzero integers. In DIMACS format the sequences representing different clauses are separated by 0’s. So for example the sequence

\[
1\ 2\ 3\ 0\ -1\ -2\ 0\ -1\ 3\ 0\ -2\ -3\ 0
\]

represents the cnf \([\{X_1,X_2,X_3\},\{\neg X_1,\neg X_2\},\{\neg X_1,X_3\},\{\neg X_2,\neg X_3\}\] in our Prolog representation. You are to write a Prolog predicate `dimacs(List,Cnf)` which given a conjunctive normal form `List` in DIMACS representation creates a Prolog representation `Cnf`. Make sure to handle correctly the cases for empty clauses and empty cnfs.

For example,

```prolog
?- dimacs([1, 2, 0, -1, 2, 3, 0],Cnf).
Cnf=[[X,Y],[-X,Y,Z]] ;
No
```

### 35 Evaluate a WFF

Write a Prolog predicate `evaluate(Wff,Value)` which given a well founded formula evaluates if it is true or false (denoted by 0,1). For example:

```prolog
?- evaluate(X+Y,V).
X = 0, Y = 0, V = 0 ;
X = 0, Y = 1, V = 1 ;
X = 1, Y = 0, V = 1 ;
X = 1, Y = 1, V = 1 ;

?- evaluate(1 -> (0 -> 1;0),V).
V = 0
?- evaluate(1 -> (0 -> 1;1),V).
V = 1
```
36  Size of a Formula

1. Write a Prolog predicate `wff_size(Wff,N)` which given a well founded formula computes the number of operators it contains. For example,

   ```prolog
   ?- wff_size(A*B+(-C),N).
   N=3
   ```

2. Write a Prolog predicate `cnf_size(Cnf,N)` which given a cnf formula computes the number of all operators it contains. For example,

   ```prolog
   ?- cnf_size([[-BX,BY,B],[BX,-BY,B]],N).
   N=7
   ```

37  Even Bits

Write a Prolog predicate `even(Bits,F)` which given a list of Bits creates a well founded formula F that states that an even number of the bits are true.

38  Sat Encoding (Order encoding)

Recall that a unary number is a sequence of 1’s followed by a sequence of 0’s. For example the number 3 in a 5 bit representation is 11100 and the number 0 in a 5 bit representation is 00000. In this question we will represent unary numbers as lists. Of course such lists might contain variables or the values 0,1.

**unary (10 points)**

Let `Xs` be a list of bits representing a unary number. You are to write a Prolog predicate `unary(Xs,Cnf)` which constructs a CNF which is satisfiable if and only if the bits in `Xs` indeed represent a unary number.

**unary greater than (10 points)**

Let `Xs` and `Ys` denote lists of bits (of the same length). You are to write a Prolog predicate `gt(Xs,Ys,Cnf)` which constructs a CNF which is satisfiable if and only if the bits in `Xs` and `Ys` represent unary numbers such that `Xs > Ys`.

39  Sat Encoding (direct encoding)

Recall that a unary number is represented as a sequence of bits in which exactly one bit takes the value 1. The position of that bit determines the value of the number represented. For example, `[0,0,1,0,0,0,0]` represents the value 3 and `[0,1,0]` the value 2. Note that also `[0,0,1,0]` represents the value 3. Of course such lists might contain variables or the values 0,1. Note that we do not represent the number 0 with this definition, but this has nothing to do with the question below.

Your task is to write a Prolog predicate `unary_diff(Xs,Ys,Cnf)` which encodes that `Xs` and `Ys` are different unary numbers.
You are given the predicate `exactlyOne(Xs, Cnf1-Cnf2)` which given a list of bits `Xs` states (as a difference list) that `Xs` is a legal number in unary representation. (You do not need to write this predicate — it is given!).

For example,

?- exactlyOne([A,B,C], Cnf-[]).
   Cnf = [[A,B,C],[¬A,¬B],[¬A,¬C],[¬B,¬C]]

?- exactlyOne([A,B], Cnf1-Cnf2).
   Cnf1 = [[A,B],[¬A,¬B]|Cnf2]

?- unary_diff([A,B,C,D],[W,X,Y,Z],Cnf), sat(Cnf).
   A=0, B=0, C=0, D=0,
   W=1, X=0, Y=0, Z=0

?- unary_diff([A,B,C,D],[0,0,0,0],Cnf), sat(Cnf).
   A=1, B=0, C=0, D=0

?- unary_diff([A,B,C,D],[X,Y,Z],Cnf), sat(Cnf).
   A=0, B=0, C=0, D=0,
   X=1, Y=0, Z=0

?- unary_diff([0,0,1,0],[0,0,1],Cnf), sat(Cnf).
   no

?- unary_diff([0,0,1],[0,1],Cnf), sat(Cnf).
   yes

40 Sat Encoding (Binary encoding)

Recall the predicate `add(Xs, Ys, Zs, Cnf)` described in class which specifies a conjunctive normal form `Cnf` involving the (lists of) propositional variables `Xs, Ys, Zs` which represent binary numbers. This `Cnf` specifies that the sum of the two binary numbers `Xs` and `Ys` is `Zs`. Recall that binary numbers are represented “least significant bit first”, so `[0,0,1]` represents the value 4. For example,

?- add([1,1,1],[0,0,1],Z,Cnf), sat(Cnf).
   Z = [1, 1, 0, 1]

?- add([1,1,1],[0,1],Z,Cnf), sat(Cnf).
   Z = [1, 0, 0, 1]

Your task is to write a Prolog predicate `times(Xs, Ys, Zs, Cnf)` which generates a a conjunctive normal form `Cnf` which expresses that `Zs` is the multiplication of `Xs` and `Ys`. You may use the predicate `add(Xs, Ys, Zs, Cnf)` specified above. For example,

?- times([1,1,1],[1,1,1],Zs,Cnf), sat(Cnf).
Zs = [1, 0, 0, 0, 1, 1, 0]

?- times([1,0,1],[1,1],Zs,Cnf),sat(Cnf).
Zs = [1, 1, 1, 1, 0, 0]
41 Sorting Networks

Let \(\text{comparator}(A,B,C,D)\) represent a (Boolean) comparator with “inputs” \(A,B\) and “outputs” \(C,D\). A comparator specifies that \(C,D\) are the same as \(A,B\) but sorted (from the higher value to the lower value). A network (of comparators) is represented as a list of comparators. A sorting network is a network of comparators which sorts its inputs. For example the following represents a 4 \(\times\) 4 sorting network (with inputs \([X_1,X_2,X_3,X_4]\) and outputs \([Y_1,Y_2,Y_3,Y_4]\)):

\[
\text{[comparator}(X_1,X_2,A_1,A_2), \text{comparator}(X_3,X_4,A_3,A_4), \text{comparator}(A_1,A_3,Y_1,B_2), \text{comparator}(A_2,A_4,B_3,Y_4), \text{comparator}(B_2,B_3,Y_2,Y_3)\text{]}\]

Let \(Cs\) be a network (not necessarily a sorting network) in which some of the arguments in some of the comparators have Boolean values. You are to write a Prolog predicate \(\text{propagate}(Cs,Cs1)\) which propagates the given values and removes (in view of these values) as many comparators as possible. For example,

\[
?\text{-} Cs = \text{[comparator}(X_1,X_2,A_1,A_2), \text{comparator}(X_3,X_4,A_3,A_4), \text{comparator}(A_1,A_3,Y_1,B_2), \text{comparator}(A_2,A_4,B_3,Y_4), \text{comparator}(B_2,B_3,Y_2,Y_3)\text{]}, \text{propagate}(Cs,Cs1).\]

\(Cs1 = \text{[comparator}(X_1,X_2,1,A_2), \text{comparator}(X_3,X_4,0,A_4), \text{comparator}(A_2,A_4,1,Y_4)\text{]}\)

\(Y_1 = 1\)

\(Y_2 = 1\)

?\text{-} Cs = \text{[comparator}(X_1,X_2,A_1,A_2), \text{comparator}(X_3,X_4,A_3,A_4), \text{comparator}(A_1,A_3,Y_1,B_2), \text{comparator}(A_2,A_4,B_3,Y_4), \text{comparator}(B_2,B_3,0,Y_3)\text{]}, \text{propagate}(Cs,Cs1).\]

\(Cs1 = \text{[comparator}(X_1,X_2,1,A_2), \text{comparator}(A_3,Y_1,0), \text{comparator}(A_1,1,Y_1)\text{]}\)

\(Y_3 = 0\)

\(Y_4 = 0\)

?\text{-} Cs = \text{[comparator}(X_1,X_2,A_1,A_2), \text{comparator}(X_3,X_4,A_3,A_4), \text{comparator}(A_1,A_3,Y_1,B_2), \text{comparator}(A_2,A_4,B_3,Y_4), \text{comparator}(B_2,B_3,Y_2,1)\text{]}, \text{propagate}(Cs,Cs1).\]

\(Cs1 = \text{[comparator}(X_1,X_2,1,A_2), \text{comparator}(X_3,X_4,1,A_4), \text{comparator}(A_2,A_4,1,Y_4)\text{]}\)

\(Y_1 = 1\)

\(Y_2 = 1\)