1 2001 Paper 5 Question 7

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2001p5q7.pdf

Consider the following Prolog program, which is intended to define the third argument to be the maximum value of the first two numeric arguments:

```
max(X, Y, X) :- X >= Y, !.
max(X, Y, Y).
```

(a) Provide an appropriate query to show that the above program can give an incorrect result.

The query below returns true – when it's clearly false.

```
max(2, 1, 1).
```

(b) Explain the cause of the error.

The second clause has no guard. So if the first condition is not satisfied then the second will always be satisfied – even if it should not be. So any false condition of the form `max(X, Y, Y)` will return true.

this is about unification failing and causing fall-though.

(c) Suggest a correction.

```
max2(X, Y, X) :- X >= Y.
max2(X, Y, Y) :- X < Y.
```

yeah, but now we sometimes have to do two tests, which is slower. is there another fix that only does the test once (and addresses the root cause)?

```
max(X,Y,Z) :- X >=Y, !, Z=X.
max(X,Y,Y).
```

(d) Write a Prolog program to find the maximum of a list of numbers.

```
maxlist([H], H).
maxlist([H|T], X) :- maxlist(T, Y), max2(H, Y, X).
```

does this find all the equal-max values under backtracking?

2 1996 Paper 6 Question 7

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y1996p6q7.pdf

(a) Describe how lists that are represented by difference lists may be concatenated (or "appended") in constant time.

Difference lists are lists with exposed pointers at the end. Empty difference lists are represented by

```
A-A.
```

Appending to difference lists can be done by the single fact

```
append(A-B, B-C, A-C).
```

(b) Define a procedure `rotate(X, Y)` where both `X` and `Y` are represented by difference lists, and `Y` is formed by rotating `X` to the left by one element.

```
% rotate(+[H|T]-[H|X], ?T-X)
% succeeds if the second argument is the first but rotated
rotate([H|T]-[H|X], T-X).
```

3 1997 Paper 6 Question 7

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y1997p6q7.pdf

A binary tree is constructed from binary compound terms `n(a, b)` called *nodes*, where components `a` and `b` are either nodes or integers. Suppose integer components are restricted to the values 0 and 1.

Write a Prolog program to return a list of all the 0's and a list of all the 1's in a given tree. For example, the goal `enum(n(n(0,1),1),X,Y)` should instantiate `X` to `[0]` and `Y` to `[1, 1]`. The program is required to use difference lists.

```
enum(T, X, Y) :- enum2(T, X-[], Y-[]).
enum2(0, [0|X]-X, Y-Y).
enum2(1, X-X, [1|Y]-Y).
enum2(n(A, B), X1-X3, Y1-Y3) :- enum2(A, X1-X2, Y1-Y2), enum2(B, X2-X3, Y2-Y3).
```

✓

>= and =< evaluate both sides -- they do proper integer comparisons

```
maxlist([H], H).
maxlist([H|T], Y) :- maxlist(T, Y), Y >= H.
maxlist([H|T], H) :- maxlist(T, Y), !, H >= Y.
```

The best way to use this is take -- think generate-and-test!
This whole mess came about because we tried to do it in an imperative way.

Constraint satisfaction paradigm

- accumulate a large set of constraints and try to satisfy them in any order
- as the data becomes available in any order
- constraint satisfaction terminates earlier than generate-and-test since you don't have to generate the WHOLE data structure.

TODO use the constraint satisfaction library (CLP)

- ie solve sudoku + fukoshiki

Shaun Holden glosses over "where does the heuristic come from"