Sample exam AI Languages

General remarks

- All exercises concern PROLOG
- All exercises will be judged on a scale from 1-10
- Hand in the questions together with your answers
- The exam is **not** an "open-book" exam; so you may not use books, readers, or notes.
- You may assume that commonly-used predicates (like the list predicates **`insert`**, **`del`**, **`member`**, etc.) are pre-defined. However, clearly indicate the predicates that you assume to be predefined.

1. Give a PROLOG program **`range(M, N, Ns)`** to generate a list of integers **`Ns`** in the range from **`M`** to **`N`**.

2. Define a PROLOG program **`split(Numbers, Positives, Negatives)`**, that splits a given list **`Numbers`** of integers into two new lists, the list of positive numbers (0 inclusive) **`Positives`** and the list of negative numbers **`Negatives`**. E.g., **`split([3,-6,5,0,-2], Positives, Negatives)`** holds if **`Positives = [3,5,0]`** and **`Negatives = [-6,-2])`**. Use cuts to make the program efficient.

3. Give a PROLOG predicate **`disjoint(L1, L2)`**, that is true if **`L1`** and **`L2`** are two lists without common elements.

4. The Ackermann function is defined as follows:

   ackermann(0, *N*) = *N* + 1;
   ackermann(*M*, 0) = ackermann(*M* – 1, 1);
   ackermann(*M*, *N*) = ackermann(*M* – 1, ackermann(*M*, *N* – 1)).

   Give a PROLOG implementation of the Ackermann function.

5. Define a PROLOG program **`flatten( Xs, Ys)`**, that holds if **`Ys`** is the list with elements that occur in the list of lists **`Xs`**. The elements of **`Xs`** can be elements themselves or can be lists, so elements in **`Xs`** may be nested arbitrarily deep. As an example: the goal **`flatten( [[a], [b], [c,a], [[b]]], [a,b,c,a,b])`** is true.
   Hint: you best can use the standard function **`atomic(X)`**, that is true if **`X`** is an atom or a number.

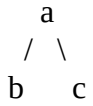
6. Give a PROLOG program **`subsum( Set, Sum, SubSet)`** such that with **`Set`** a list of numbers, **`SubSet`** is a sublist of this list, and such that the sum of the numbers in **`SubSet`** equals **`Sum`**. For eample:

```
?- subsum([1,2,5,3,2], 5, Sub).
Sub = [1,2,2];
Sub = [2,3];
Sub = [5];
Sub = [3,2];
no.
```

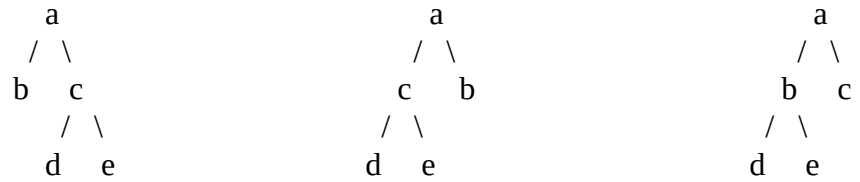
7. Give a PROLOG program **strangeSort** that sorts lists of positive integers as follows: the first element of the sorted list is the smallest element, the second the largest element, the third the smallest-but-one, the fourth the largest-but-one, and so on. For example, given the list **[3,6,1,2,4,7,5]** the program should yield the sorted list **[1,7,2,6,3,5,4]**. You may choose the method to do this yourself.

8. Write a PROLOG program **uniqueMerge** that merges 2 sorted lists in sorted order, but omits duplicates. You may assume that the two input lists are sorted, and don't contain duplicates themselves. So: **uniqueMerge([2,3,5,7,8], [1,2,5,6,8], L)** is true if and only if **L = [1,2,3,5,6,7,8]**.

9. Suppose we represent a binary tree with the ternary functor **tree(Element, Left, Right)**, in which **Element** is the root element, and **Left** and **Right** the left and right subtrees. Th empty tree is represented by the atom **void**. The tree



   then is represented as **tree(a, tree(b, void, void), tree(c, void, void))**. Write a PROLOG program **binary_tree(Tree)** that determines if the argument **Tree** is a binary tree. Next give a program **tree_member(Element, Tree)** that determines if **Element** is an element in the tree **Tree**. Finally, write a program **isotree(Tree1, Tree2)**, that determines whether 2 trees **Tree1** and **Tree2** are isomorphous. [Two trees T1 and T2 are isomorphous if T2 can be obtained from T1 by re-ordering the branches. For example, in the next figure the first two trees are isomorphous; the third one not.]



10. Use **bagof** to implement a relation **cycles_of(Graph, Cycles)**, that determines all cycles **Cycles** of a given graph **Graph**. A graph is represented as a list of edges, for example **[a-b, b-c, c-a, b-d, d-c]**.

Good luck.