Prolog Project Report T12G03

Main predicates and the flow of their helpers:

- **a) straight_chain_alkane(N,A)**
  - **straight_chain_helper(N,A,T) -helper-**
- **b) branched_alkane(N,BA)**
  - **length_chain(N,X,Y)**
    - generate(S,E,X)
    - length_chain(N,K,X,Y)
  - **branched_alkane(X,Y,BA) -helper-**
    - straight_chain_alkane(N,A)
      - straight_chain_helper(N,A,T)
    - break_down(N,L)
      - break_down(N,X,L)
        - generate(S,E,X)
    - permutate_list_remove(A,L,X)
      - permutate_list([H|T],List,R)
        - permutate(Val,List,R)
          - generate(S,E,X)
          - edit_list(Pos,Val,List,R)
            - add_branch_to_carbon(InC, BSize, ResC)
              - branch_name(S,N)
                - atomic_list_concat([c,S,h,HS],N)
- **c) isomers(N,[A2|A])**
  - **duplicate_remove([H|T],[H|A]) -helper-**
  - **reverse_middle(List,R) -helper-**

Predicts and a brief description:

- **straight_chain_alkane(N,A)**
    - is responsible for generating a straight chain alkane, it checks if N = 1 it returns carb(h,h,h,h) else it generates the longest possible chain by calling its helper straight_chain_helper(N,A,T)
- **straight_chain_helper(N,A,T)**
    - this is a recursive call where A acts as an accumulator to follow up the list ,after each addition of an atom and when N = 1 the call will append the last atom carb(h,h,h,c) and return the chain as list T.
    - handles the condition ("No branch should be attached to either carbon atom at the end of the chain") as it adds the last element always as carb(_,h,h,h).
- **branch_name(S,N)**
    - explained in the notes
- **add_branch_to_carbon(InC, BSize, ResC)**
    - explained in the notes
    - handles the condition ("a carbon atom is allowed to have a down branch attached") by checking that either top is h so it modifies the top or the top is not h so it modifies the bottom.
- **generate(S,E,X)**
    - generates numbers starting from S to E and returns every integer between them in X.
- **break_down(N,L)**
    - used to find every different possibility that can number N be broke to in a form of list L, it calls its helper break_down(N,X,L).

- **break_down(N,X,L)**
    - helper of break_down(N,L) where X is used to keep track of the last number added as head to the previous list, to avoid having duplicates and to get a sorted list
    - handles the condition ("If a carbon atom has two branches attached to it, then the number of carbon atoms in the top branch has to be less than or equal to the number of atoms in the down branch") as the list is always sorted, therefore smaller values are always added upwards
- **edit_list(Pos,Val,List,R)**
    - receives a single element value from the list generated by break down as Val, and edits the element in Pos in the List by calling add_branch_to_carbon(List[Pos], Val, Result)
    - it handles the condition ("The number of carbon atoms in any branch should not lead to any new chains that are longer than the current longest chain") by checking the left and right of every atom before editing it, making sure that the new branch will not be greater than the longest branch.
- **permutate(Val,List,R)**
    - generates different permutations of edit_list(Pos,Val,List,R) by calling generate for different values of Pos starting from 1 till List length.
- **permutate_list(K,List,R)**
    - generates different values of permutate(Val,List,R) by traversing a list K and calling Val for each value of K.
- **permutate_list_remove(A,L,X)**
    - calls permutate_list(K,List,R) and removes the duplicates.
- **length_chain(N,X,Y)**

- receives N and return the value of the next longest chain X and the number Y to be used in break_down, calls generate K to get a number from 1 to N and calls its helper length_chain(N,K,X,Y).

- **length_chain(N,K,X,Y)**
  - get K and returns X as N-K and Y as K if X>2.
  - handles case that lowest branched alkane is having 3 carbons in the center chain.
- **branched_alkane(N,BA)**
  - uses length_chain(N,X,Y) and calls its helper branched_alkane(X,Y,BA) returning the branched alkane
- **branched_alkane(X,Y,BA)**
  - takes X as the length of the center chain and Y to be break_down the calls permutate_list_remove to get all possible permutation returned as BA
- **isomers(N,[A2|A])**
  - makes a set of all possible results for branched alkane with value N then calls remove duplicates , more over it adds the straight chain in the begging of the list.
- **duplicate_remove([H|T],Result)**
  - takes a list [H|T] and checks if the reverse of H is repeated in the rest of the list it removes H else it keeps H and checks the rest of the list T.
- **reverse_middle(List,R)**
  - skips the first element and last element in a List and reverses all other elements inside this list and returns the result as R