**Prolog project**

**Submitted by:**

Esraa Salah 31-6542

Nourhan Ramadan 31-5601

**Main predicates:**

**straight\_chain\_alkane(N,L) :** This predicate generates straight alkane containing N carbon atoms.

**branched\_alkane(N,BA) :**

This predicate generates the branched alkane structure as follows:-

1-Getting all lengths of possible main chains in the branched alkane(using possible\_length(N,MiniN)).

2-We get the number of remaining carbon atoms to be put in form of branches (Bsize).

3- Generating different configures for the branches (using break\_down(Bsize,Branches).

4-Generating the main chain (using straight\_chain\_alkane(MiniN,MainChain).

5- Generating the list Atoms which contains the possible positions (indices) that the branches could be added to without forming a branch having a length greater than the length of the longest chain (using where\_to\_add\_branch(MiniN, Banches, Atoms)).

6-We generate the final branched alkane by adding the branches to the main straight chain at (Atoms) that identify the positions of carbon atoms where branches are to be added (using add\_branch\_to\_straight\_chain(Branches, Atoms, Mainchain, BA)).

**isomers(N,A) :**

This predicate is used to filter more the branched alkanes generated,as follows:-

1-We generate all branched alkanes without repititions (using setof(BA, N^branched\_alkane(N, BA), Branched)).

2-We add the straight chain alkane to the list.

3-We filter the list by removing all the repeated mirrors across y-axis (using filter(Iso, A)).

**Helper predicates:**

**branch\_name(S, N) :** It generates the name of the branch with N carbon atoms.

**create\_list(S, N, L) :** It creates a list of numbers starting from S till N.

**possible\_length(N, MiniN) :** Finds all possible lengths of branched alkanes of size N.

**add\_branch\_to\_carbon(InC, BSize, ResC)** : It creates the new term of the carbon atom, after adding the branch of Bsize carbon atoms to it.

**break\_down(N, L) :** Form a sorted list of all possible combinations of integers that sums to N.

**add\_branch\_to\_straight\_chain(Branches, Position, OldChain, NewChain) :** It adds the generated branches to the given positions in the longest chain .

**get\_carbon(L, N, C) :** This predicate gets the carbon with index N from the alkane list.

**replace\_carbon(L, N, L1, C) :** This predicate replaces the atom in the list with index N with the atom C.

**where\_to\_add\_branch(Nlongest, Lbranches, Latoms) :**This predicate produces a list with all the possible locations in the longest chain to add a branch while making sure that the positions contained in the generated list will not produce a list with a length greater than the length of the longest chain or a carbon atom with more than two branches.

**check\_number(L1, L2) :** This predicate checks if each position generated in the list of positions is not repeated more than twice as each carbon can’t hold more than two branches.

**count(L1, L2, N) :** It counts the number of occurrences of the head of L1 in L2.

**check\_longest(Nlongest, Lbranches, LAtoms) :** This predicate checks if putting a branch on a certain atom would generate a longer chain than the longest chain.

**nearest\_end(N, Atom, Dis) :** This rule returns the distance to the nearest end of a list, whether the head of the list or the last element in the list.

**minimum(X, Y, Z) :** This predicate finds the minimum number of two numbers.

**create\_atoms(N, Latoms, Lpossible) :** This rule creates a list of all the possible locations where a branch could be added without checking any conditions regarding the length of the generated list.

**filter(L1, L2) :** This rule takes a list L1 and generates a mirror of each list in the list L1 across y-axis , removes all its mirrors from L1 and returns the list L2 without any mirrors of any of the elements in list L1.

**mirror(L1, L2) :** This rule generates a mirror list of a given branched alkane across y-axis.

**add\_to\_end(X, L,Res) :** This rule adds an atom to the end of a list.