Donovan Lee 016741645

Brian Tran 016678231

HW 5: Puzzling Prolog

**KenKen Queries**:

%solving the KenKen Puzzle

S = [

[\_,\_,\_,\_,\_,\_],

[\_,\_,\_,\_,\_,\_],

[\_,\_,\_,\_,\_,\_],

[\_,\_,\_,\_,\_,\_],

[\_,\_,\_,\_,\_,\_],

[\_,\_,\_,\_,\_,\_]

],

Cages = [

cage(div, 2, [[0, 0], [1, 0]]),

cage(mult, 180, [[0, 1], [0, 2], [1, 1], [1, 2]]),

cage(mult, 12, [[0, 3], [1, 3], [2, 3]]),

cage(sub, 2, [[0, 4], [0, 5]]),

cage(id, 3, [[2, 0]]),

cage(mult, 30, [[2, 1], [2, 2], [3, 2]]),

cage(mult, 360, [[2, 4], [3, 3], [3, 4], [4, 3]]),

cage(add, 13, [[1, 4], [1, 5], [2, 5], [3, 5]]),

cage(mult, 20, [[4, 0], [3, 0], [3, 1]]),

cage(sub, 5, [[4, 4], [4, 5]]),

cage(mult, 144, [[5, 0], [5, 1], [4, 1], [4, 2]]),

cage(add, 13, [[5, 2], [5, 3], [5, 4], [5, 5]])

],

solve(S,Cages).

sum\_list([1,3,5],S).

%S = 9%

product\_list([1,3,5], S).

%S = 15%

adds\_to(1,2,3).

mults\_to(1,2,2).

%test puzzle

S = [

[1,2],

[3,4],

[5,6],

[1,2],

[3,4],

[5,6]

],

%get\_cell

get\_cell(S,[0,0],Val).

%check cages

check\_cages(S, [cage(add,10,[[0,0],[0,1]]), cage(add,3,[[1,0],[1,1]])]).

%cell\_value

cell\_values()

%add

check\_constraint(S,cage(add,X,[[0,0],[0,1]])).

%mult

check\_constraint(S,cage(mutl,X,[[0,0],[0,1]])).

%sub

check\_constraint(S,cage(sub,X,[[0,1],[0,0]])).

%div

check\_constraint(S,cage(sub,X,[[0,0],[0,1]])).

%id

check\_constraint(S,cage(id,X,[[1,1]])).

**Code**:

:- use\_module(library(clpfd)).

*% sum and product not really used in this program.*

sum\_list([],0).

sum\_list([H|T], S) :- sum\_list(T,D), S is D + H.

product\_list([],0).

product\_list([H],H).

product\_list([H|T], S) :- product\_list(T,D), S is D \* H.

cage([],[],[]). *%atom, target value, list of cell cords*

*%cell values*

get\_cell(S,[I,J],Val) :-

nth0(I,S,Elem),

nth0(J,Elem,Return),

Val #= Return.

cell\_values(Cells, S, Values) :-

maplist(get\_cell(S), Cells, Values).

*%validating cages*

*%*

check\_constraint(S, cage(id, Value, Cells)) :-

cell\_values(Cells, S, Values),

nth0(0, Values, A),*%grabs the index values*

Value #= A.*%returns value itself*

check\_constraint(S, cage(add, Value, Cells)):-

cell\_values(Cells, S, Values),

foldl(adds\_to, Values, 0, Result),*%recursively adds*

Value #= Result.

check\_constraint(S, cage(mult, Value, Cells)):-

cell\_values(Cells, S, Values),

foldl(mults\_to, Values, 1, Result),*%recursively multiplies must be 1 accumulator cant be 0 else it multi by 0*

Value #= Result.

*%Does the subtraction operation of the values of the cells after the conversion from the cell coordinates, uses the or operator to choose which values to return from*

check\_constraint(S, cage(sub, Value, Cells)) :-

cell\_values(Cells, S, Values),

nth0(0, Values, A),

subs\_to(A, B, RetVal\_1),

nth0(1, Values, B),

subs\_to(B, A, RetVal\_2),

(Value #= RetVal\_1 ; Value #= RetVal\_2).

check\_constraint(S, cage(div, Value, Cells)) :-

cell\_values(Cells, S, Values),

nth0(0, Values, A),*%looks for index*

divs\_to(A, B, RetVal\_1), *%does the div function to get value*

nth0(1, Values, B),

divs\_to(B, A, RetVal\_2),

(Value #= RetVal\_1; Value #= RetVal\_2).

*%check one cage to see if all the constraints are satisfied(true),*

*%if so, then the cage is true. Put this functor inside maplist.*

check\_cages(S, Cages):-

maplist(check\_constraint(S),Cages).

*%solving the game*

solve(S, Cages) :-

*% S must have 6 rows*

length(S,6), *%checks for length of 6*

*% Each row in S must be length 6*

transpose(S, Rows), *%transposes the S matrix*

length(Rows, 6), *% checks to see if Rows has length of 6*

*% Each row in S must only contain values of 1 to 6*

append(S,Values),*%Values is a list that will append all values of S*

Values ins 1..6, *%checks if list contains values btw 1-6*

*% The entries in S must satisfy the cages of the puzzle*

check\_cages(S,Cages),

*% Each row in S must contain all distinct values(no dups)*

maplist(all\_different,S),

*%Each column in S must contain all distinct value*

transpose(Rows,Columns), *%flip rows back to columns to check*

maplist(all\_different, Columns),

maplist(label,S).

*%sums | products | sub | divs*

adds\_to(X,Y,Z):- Z #= Y + X.

mults\_to(X,Y,Z):- Z #= Y \* X.

subs\_to(X,Y,Z):- Z #= Y - X.

divs\_to(X,Y,Z):- Z #= Y//X.