**X. Prolog implementation**

We have implemented the described graph transformation in a simple Prolog program.

Predicate *collapse(Nodes,Edges,NewNodes,NewEdges)* defines when a topology (represented by a list of *Nodes* and a list of *Edges*) can be reduced to a new topology (*NewNodes*,*NewEdges*) by collapsing two nodes into a new node:

collapse(Nodes,Edges,NewNodes,NewEdges) :-

reducible(U,V,D,Z,Edges),

collapseNodes(U,V,D,Z,N,Nodes,NewNodes),

collapseEdges(U,V,N,Edges,NewEdges).

Predicate *reducible(U,V,h,Z,Edges)* is defined by two clauses that directly implement rules (rv) and (rh) of Definition 8:

reducible(U,V,v,\_,Edges) :-

member(edge(v,U,V),Edges),

\+ member(edge(v,\_,U),Edges),

findall(X, (path(v,X,V,Edges),(path(h,U,X,Edges);path(h,X,U,Edges))), []).

reducible(U,V,h,Z,Edges) :-

member(edge(h,U,V),Edges),

\+ member(edge(v,\_,U),Edges),

\+ member(edge(v,\_,V),Edges),

member(edge(v,U,Z),Edges),

member(edge(v,V,Z),Edges).

where predicate *path(D,X,Y,Edges)* defines whether there is a path from node *X* to node *Y* (consisting of vertical or horizontal edges, depending on *D*):

path(D,X,Y,Edges) :-

path(D,X,Y,[],Edges).

path(D,X,Y,\_,Edges) :-

member(edge(D,X,Y),Edges).

path(D,X,Y,Visited,Edges) :-

member(edge(D,X,W),Edges),

W \== Y, \+ member(W, Visited),

path(D,W,Y,[W|Visited],Edges).

Predicate *collapseNodes(U,V,D,Z,N,Nodes,NewNodes)* defines when a set of nodes *NewNodes* is the result of collapsing (vertically or horizontally, on *D*) two nodes *U* and *V* from a set of *Nodes*:

collapseNodes(U,V,D,Z,N,Nodes,[NewNode|NewNodes]) :-

deleteNodes([node(U,\_,Uc),node(V,Vt,Vc)],Nodes,NewNodes),

N = m(U,V,D),

( (D==v, NewNode = node(N,Vt,l(Vt,v,Uc,Vc))) ;

(D==h, member(node(Z,Zt,\_),NewNodes), NewNode = node(N,Vt,l(Zt,h,Uc,Vc))) ).

where predicate *deleteNodes([X,Y],Nodes,NewNodes)* simply defines when a set of nodes *NewNodes* is the result of deleting two nodes *X* and *Y* from a set of *Nodes*:

deleteNodes([X,Y],[X|L],NewNodes) :-

deleteNode(Y,L,NewNodes).

deleteNodes([X,Y],[Y|L],NewNodes) :-

deleteNode(X,L,NewNodes).

deleteNodes([X,Y],[Z|L],[Z|NewNodes]) :-

X \== Z, Y \== Z, deleteNodes([X,Y],L,NewNodes).

deleteNode(X,[X|L],L).

deleteNode(X,[Y|L],[Y|NewNodes]) :-

X \== Y, deleteNode(X,L,NewNodes).

Predicate *collapseEdges(U,V,Edges,NewEdges)* defines when a set of edges *NewEdges* is the result of collapsing a set of *Edges* as a consequence of collapsing two nodes *U* and *V*:

collapseEdges(U,V,N,Edges,NewEdges) :-

collapseEdges2(U,V,N,Edges,NewEdgesWithDoubles),

((NewEdgesWithDoubles==[],NewEdges=[]) ;

(setof(edge(DD,X,Y),member(edge(DD,X,Y),NewEdgesWithDoubles),NewEdges))).

collapseEdges2(\_,\_,\_,[],[]).

collapseEdges2(U,V,N,[edge(\_,X,Y)|Edges],NewEdges) :-

( (X==U,Y==V);(X==V,Y==U) ),

collapseEdges2(U,V,N,Edges,NewEdges).

collapseEdges2(U,V,N,[edge(D,X,Y)|Edges],[edge(D,NewX,NewY)|NewEdges]) :-

( (X \== U, X \== V, (Y==U;Y==V), NewX=X, NewY=N) ; ((X==U;X==V), Y \== U, Y \== V, NewX=N, NewY=Y) ),

collapseEdges2(U,V,N,Edges,NewEdges).

collapseEdges2(U,V,N,[edge(D,X,Y)|Edges],[edge(D,X,Y)|NewEdges]) :-

X \== U, X \== V, Y \== U, Y \== V,

collapseEdges2(U,V,N,Edges,NewEdges).

Finally, predicate *loop/1* is used to reduce a topology to a single node. After uploading the set of *Nodes* and *Edges* forming the topology, and checking that the topology is well-formed, it exploits predicate *loop/3* to collapse pairs of nodes, one at a time:

loop(Result) :-

findall(node(N,T,C), node(N,T,C), Nodes),

findall(edge(D,N1,N2), edge(D,N1,N2), Edges),

wellFormedTopology(Edges),

loop(Nodes,Edges,Result).

wellFormedTopology(Edges) :-

findall(X, ( member(edge(h,X,X),Edges); path(v,X,X,Edges) ), []), %def 3.(i)

findall(X, ( member(edge(v,X,Y),Edges), member(edge(v,X,Z),Edges), Y\==Z ), []). %def 3.(ii)

loop([node(\_,\_,Nc)],[],Nc).

loop([Node1,Node2|Nodes],Edges,Result) :-

collapse([Node1,Node2|Nodes], Edges, NewNodes,NewEdges),

loop(NewNodes,NewEdges,Result).