Hyper Graph Module API

This document specifies and describes the API for a Generic and Extensible Hyper Graph Module. The three fundamental classes for this module are *graph*, *graph-vertex* and *graph –edge*. Unless otherwise noted all symbols are in the hypergraph package

# Graph API

### (MAKE-GRAPH &key <name><class>)

This creates and returns a graph object. If <name> is specified it should be a string and it will be retrievable via *find-graph*, otherwise the graph will have no name and will not be retrievable via *find-graph*.

### (FIND-GRAPH <name>)

Returns the graph object named <name> if such a graph exists.

### (DELETE-GRAPH <graph>)

Deletes the graph specified by <graph> if such a graph exists. The argument <graph> can be a graph object or a string naming a graph.

### (PRINT-GRAPH <graph>)

This function prints a list of the vertices, the edge-types and the edges of the specified graph object.

### (SAVE-GRAPH <graph> <mode> <destination>)

This function saves the graph specified by <graph> to a file using the adjacency list representation of a graph. Currently the only supported modes are :lisp :lisp2 and <destination> is expected to be a file or path name.

### (RESTORE-GRAPH <mode> <source> &key <graph><graph-class><vertex-class><edge-class>)

This function restores the graph described by <source> and returns an object of class *graph*. It is assumed that an adjacency list representation of the graph is contained in <source>. Current ly the only supported <mode> is :lisp and <source> is expected to be a file-name.

### (MAP-GRAPH-VERTICES <graph> <function>)

This function applies <function> to all vertices of <graph>. The argument <function> should be a function of one argument that will be invoked with a vertex object.

### (MAP-GRAPH-EDGES <graph> <function>)

This function applies <function> to all edges of <graph>. The argument <function> should be a function of one argument that will be invoked with an edge object.

# Graph Vertex API

### (MAKE-GRAPH-VERTEX <vertex-name> <graph> &key <class><weight>)

This function creates and returns a vertex object and adds it to specified graph in such a way that it can be retrieved by name from the specified graph. The argument <vertex-name> must be a string and the argument <graph> must be a graph object.

### (ENSURE-GRAPH-VERTEX <vertex-name> <graph> &key <class><weight>)

This function creates and returns a vertex object and adds it to specified graph in such a way that it can be retrieved by name from the specified graph. The argument <vertex-name> must be a string and the argument <graph> must be a graph object.

### (FIND-GRAPH-VERTEX <vertex-name> <graph>)

This function returns the vertex object named <vertex-name> or nil if no such vertex in the graph specified by <graph>.

### (DELETE-GRAPH-VERTEX <vertex> <graph>)

This functions deletes the vertex specified by <vertex> from the graph specified by <graph>. All inbound and outbound edges connected to <vertex> are alse deleted from the graph.

**(VERTEX-GRAPH <vertex>)**

**(VERTEX-NAME <vertex>)**

**(VERTEX-WEIGHT <vertex>)**

These three functions respectively return the name, weight and graph of the vertex specified by <vertex>. The first are readers only, whereas *vertex-weight* is setf-able.

**(FIND-OUTBOUND-VERTICES <vertex> <edge-type><graph>)**

**(FIND-INBOUND-VERTICES <vertex> <edge-type><graph>)**

These two functions respectively return the outbound and inbounds vertices that the respectively the targets and sources of the edges of type <edge-type> that connected to <vertex>

# Graph Edge API

**(MAKE-GRAPH-EDGE <edge-type><source><target><graph> &key <class><weight>)**

**(MAKE-DEFAULT-EDGE-NAME <edge>)**

**(FIND-GRAPH-EDGE <source> <edge-type> <target> <graph> &key <name>)**

**(ENSURE-GRAPH-EDGE <source> <edge-type> <target> <graph>)**

**(DELETE-GRAPH-EDGE <edge> <graph>)**

**(EDGE-NAME <edge>)**

**(EDGE-WEIGHT <edge>)**

**(EDGE-SOURCE <edge>)**

**(EDGE-TARGET <edge>)**

**(EDGE-GRAPH <edge>)**

**(FIND-OUTBOUND-EDGES <source-vertex> <edge-type> <graph>)**

**(FIND-INBOUND-EDGES <target-vertex> <edge-type> <graph>)**

**(FIND-EDGES-OF-TYPE <edge-type> <graph>)**

**(MAP-OUTBOUND-EDGES <source> <edge-type> <function>)**

**(MAP-INBOUND-EDGES <target> <edge-type> <function>)**

## Graph Edge Tables

**(MAKE-EDGE-TABLES <graph>)**

**(FIND-EDGE-TABLE <graph> <edge-type>)**

**(DELETE-EDGE-TABLES <graph> <edge-type>)**

**(EMPTY-EDGE-TABLES-P <edge-tables>)**

## Save & Restore

**(SAVE-GRAH <graph> <format> <filename> &key <metadata>)**

**(RESTOTRE-GRAPH <format> <filename> &key < graph> <graph-class> <vertex-class> <edge-class>)**

## Paths

**(MAKE-GRAPH-PATH <graph> <edges>)**

**(COMPUTE-PATH-COST <path>)**

## Subgraphs

**(MAP-NODES-IN-SUBGRAPH <vertex> <function> &key <depth> <children-accessor>)**

**(COUNT-NODES-IN-SUBGRAPH <vertex> &key <children-accessor>)**

**(PRETTY-PRINT-SUBGRAPH <vertex> &key <children-accessor>)**