Strongly Connected Components

If a CBD model's dependency graph contains dependency *cycles*, these need to be identified and replaced by an *implicit* solution (analytical or numerical). Note how often, a small Delay (or Integrator) is inserted to "break the loop" and hence avoid implicit solving. Finding dependency cycles is also known as locating *strongly connected components* in a graph. A strongly connected component is a set of nodes in a graph whereby each node is reachable from each other node in the strongly connected component.

```
# Produce a list of strong components.
# Strong components are given as lists of nodes.
# If a node is not in a cycle, it will be in a strong
# component with only itself as a member.

def strongComp(graph):

  # Do a topological ordering of nodes in the graph
  topSort(graph)

  # note how the ordering information is not lost
  # in subsequent processing and will be used during
  # Time Slicing simulation.

  # Produce a new graph with all edges reversed.
  rev_graph = reverse_edges(graph)

  # Start with an empty list of strong components
  strong_components = []

  # Mark all nodes as not visited
  # setting the stage for some form of dfs of rev_graph
  for node in rev_graph:
    node.visited = FALSE

  # As strong components are discovered and added to the
  # strong_components list, they will be removed from rev_graph.
  # The algorithm terminates when rev_graph is reduced to empty.
  while rev_graph != empty:

    # Start from the highest numbered node in rev_graph
    # (the numbering is due to the "forward" topological sort
    # on graph
    start_node = highest_orderNumber(rev_graph)

    # Do a depth first search on rev_graph starting from
    # start_node, collecting all nodes visited.
    # This collection (a list) will be a strong component.
    # The  dfsCollect() is very similar to strongComp().
    # It also marks nodes as visited to avoid infinite loops.
    # Unlike strongComp(), it only collects nodes and does not number
    # them.
```

```
component = dfsCollect(start_node, rev_graph)

# Add the found strong component to the list of strong components.
strong_components.append(component)

# Remove the identified strong component (which may, in the limit,
# consist of a single node).
rev_graph.remove(component)
```