Formal Specification of Abstract Datatypes
Exercise 5 (June 29)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

The result is to be submitted by the deadline stated above via the Moodle interface as a .zip or .tgz file which contains

- a PDF file with
  - a cover page with the title of the course, your name, Matrikelnummer, and email-address,
  - the formal specifications in the style of "Thinking Programs",
  - the CafeOBJ specifications,
  - comprehensive tests of the CafeOBJ specifications (sample reductions),
  - optionally any explanations or comments you would like to make;
- the CafeOBJ (.mod) file(s) of the specifications.

Exercise: Directed Graphs

In set theory, a *directed graph* is a pair $G = (N, E)$ where N is a set of elements called *nodes* and E is a set of pairs of nodes called *edges*.

The goal of this exercise is to specify an abstract datatype of directed graphs with associated operations where $N = \mathbb{N}_n = \{0, ..., n-1\}$ for some $n \in \mathbb{N}$.

Specification First develop a formal specification of the following shape:

```
spec GRAPH import NAT :=
... {
  type Graph = g(Nat,Edges)
  type Edges = ...
  ...
}
then ... {
  empty: Nat → Graph
  link: Graph × Node × Node → Graph

  nodes: Graph → Nat
  edges: Graph → Nat

  adjacent ⊆ Graph × Node × Node
  connected ⊆ Graph × Node × Node
  distance: Graph × Node × Node → Nat

  complete ⊆ Graph
  connected ⊆ Graph
  cyclic ⊆ Graph

  ...
}
```

Make sure that two graphs are equal if they have the same nodes and the same edges.

Here $graph(n)$ denotes the graph with n nodes and no edges while $link(g, n_1, n_2)$ denotes the graph that adds to graph g an edge from n_1 to n_2 (if n_1 or n_2 is not a node of g or g already contains that edge, the graph remains unchanged). Furthermore, $nodes(g)$ denotes the number of nodes in g while $edges(g)$ denotes the number of its edges.

The predicate $adjacent(g, n_1, n_2)$ holds if g has an edge from n_1 to n_2 (thus it does trivially not hold, if n_1 or n_2 is not a node of g). The predicate $connected(g, n_1, n_2)$ holds if there exists for some $k \geq 1$ a path of nodes $n_1 = m_1, m_2, \ldots, m_k = n_2$ where every pair m_i and m_{i+1} with $i < k$ is adjacent (thus every node n is trivially connected to itself by the path $n = m_1 = n$). For two connected nodes n_1 and n_2 in g, $distance(g, n_1, n_2)$ denotes the number of edges in the shortest path from n_1 to n_2 (thus every node has distance 0 to itself).

The predicate *complete*(g) holds if every pair of nodes in g is adjacent. The predicate *connected*(g) holds if every pair of nodes in g is connected. The predicate *cyclic*(g) holds, if there is some node in g that is connected to itself by a path with at least one edge.

Does your specification give rise to a monomorphic datatype (justify your answer)?

CafeOBJ Implement your specification by a tight CafeOBJ module

```
module! GRAPH
{
  protecting (NAT)
  ...
}
```

Test the CafeOBJ module with several reductions. Give the input and output of each test and your interpretation of the results (do they indicate errors in your specification or not?). If your specification contains errors, use the trace facilities of CafeOBJ for debugging.

Does your CafeOBJ module denote the same datatype as the one of the previous specification? Justify your answer.