Formal Specification of Abstract Datatypes
Exercise 2 (May 16)

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: File Systems

A *file system* is hierarchical collection of *files* and in *directories* which both are denoted by structured names called *paths*.

Formal Specification First, develop a formal specification of the concepts in the style presented in the course:

```
spec FILESYSTEM import STRING :=
  free type Path = root | path(Path, String)
then free
{
  type FileSystem = empty | mkdir(FileSystem, Path)
                          | write(FileSystem, Path, String)
  ...
}
then ...
{
  ...
}
```

with the following operations

```
fun parent: Path → Path
fun base:   Path → String

exists ⊆ FileSystem × Path
isdir  ⊆ FileSystem × Path
read:    FileSystem × Path → String
remove:  FileSystem × Path → FileSystem
```

The specified entities shall have the following intended interpretations:

- A (file/directory) *path* is a structured name that is either empty (denoting the *root*) or a composition of another path (its *parent*) with a *base* name (a string identifying the file/directory relative to the parent).

  Among other operations, it is possible to extract the parent and the base name of a path.

- A *file system* is either *empty* or the result of
  - creating by an operation *mkdir* a directory at a denoted path,
  - creating/updating by an operation *write* a file at a certain path with a certain content.

  In both cases, the parent of the path (of the directory/file) must already exist. In a parent directory, there cannot exist two entities (directories and/or files) with the same base name.

  Among other operations, *exists* determines whether the file system has an entity with a given path, *isdir* determines whether this entity is a directory (otherwise it is a file), *read* determines the content of a file and *remove* removes an entity (directory/file) from the file system (if an entity with that path does not exist, the file system remains unchanged).

You may assume that `STRING` provides an (otherwise unspecified) type `String` for base names and file contents.

Please consider carefully in which part of the specification you declare operations and where you add axioms. Axioms in the `free` part of the specification must be formulated in the language of conditional equational logic to ensure the existence of a free (initial) interpretation. However, do not try to further constrain the entities specified in the free part in subsequent parts of the specification by additional axioms; this will make the specification inconsistent.

Are according to your specification two file systems identical, if they provide the same directories and files and file contents? Justify your answer.

CafeOBJ Second, implement the specification in CafeOBJ by a tight module:

```
module! FILESYSTEM
{
  protecting (STRING)
  signature
  {
    [ Path FileSystem ]
    ...
  }
  axioms
  {
    ...
  }
}
```

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

Are in the CafeOBJ specification two file systems identical, if they provide the same directories and files and file contents? Justify your answer.