Programming Research Group

Theorem Proving, Michaelmas 1994

Practical Assignment 1: Fun with OBJ

Due: 19 October 1994

1 Getting Started

The main purpose of this assignment is to help you get used to actually using OBJ3. First, make and connect to some appropriately named subdirectory (such as "`obj`") and use your favorite editor (Emacs is highly recommended) to prepare the following file (possibly by copying it from my directory, but modifying the first line to show the path in your directory):

```
***> file: ~goguen/ox/tp/x94/1.obj

*** The natural numbers in Peano notation
obj NATP is sort Natp .
  op 0 : -> Natp .
  op s_ : Natp -> Natp .
endo

*** lists in Lisp-like syntax
obj LIST is
  including NATP .
  sort List .
  op nil : -> List .
  op cons : Natp List -> List .
  op car_ : List -> Natp .
  op cdr_ : List -> List .
  var N : Natp .
  var L : List .
  eq car cons(N,L) = N .
  eq cdr cons(N,L) = L .
    *** two constants for later use
  op n : -> Natp .
  eq n = s s 0 .
  op l : -> List .
  eq l = cons(n, cons(s 0, cons(s s n, nil))) .
endo

*** to see what operators do in the term algebra:
red cons(n,l).
red cons(s n,l).
```

```
red cons(s n, cons(n, l)).

*** now some reductions including equations:
red car l .
red cdr l .
red car cdr l .
red car cdr cdr l .

*** some homomorphisms from LIST:
obj HOMS is
  including LIST .
  including NAT .
  op h : Natp -> Nat .
  op h : List -> Nat .
  op h´ : List -> Nat .
  var N : Natp .
  var L : List .
  eq h(0) = 0 .
  eq h(s N) = 1 + h(N) .
  eq h(nil) = 0 .
  eq h(cons(N,L)) = h(N) + h(L).
  eq h´(nil) = 0 .
  eq h´(cons(N,L)) = 1 + h´(L).
endo

*** some experiments with the homomorphisms:
red h(n).
red h(s n).
red h(l) .
red h(cons(n,l)).
red h´(l) .
red h´(cons(n,l)).
```

Here `NAT` is a built-in module providing natural numbers in their usual form, 0,1,2,... .

Typing “`obj`” at the top (Unix) level of any Sun4 or SPARC PRG or ECS workstation should call up the latest version of OBJ3, producing a cheerful banner showing today’s date, and then the OBJ prompt, “`OBJ>`”, to which you should respond by typing

```
in 1
```

to read in your file and run it; probably you will find some typos to correct. Then you can exit OBJ by typing “q” to the prompt, edit your file, reenter OBJ, have it read the revised file, ..., until done. But because OBJ is rather large, it is actually much faster not to exit OBJ, by editing your file in one window and running OBJ in another window, or if you don’t like windows, by editing in one Emacs buffer running OBJ in another.

It is just possible that a typo may thrust you into the underlying Lisp system, in which case you may see something rather frightening, like

```
Signal 11: Segmentation violation
Break.
Broken at CATCH.  Type :H for Help.
>>
```

However, if you bravely type the magical formula "`:Q`" at this point, you will be returned to the top level of OBJ. (The other options revealed by typing "`:H`" probably won't help much unless you know something about how Kyoto Common Lisp works.)

Personally, I like to run OBJ from within a shell in an Emacs window; this way, I get a trace of the computation that I can browse through using the powerful Emacs search commands; and later on, I can file it away as a log file (called, for example "`1.lg`"). Unfortunately, running OBJ in an Emacs shell on a Sun 3/50 with a small memory may be very slow. However, you could log onto some vacant booth from within your Emacs shell, and run OBJ there; then Emacs is on your machine, while OBJ is on the other, thus greatly reducing the memory requirements. Alternatively, you could pipeline OBJ's output into a file, using standard Unix commands; this only needs one machine, but it gives less editing power.

Good Luck!

2 Some Questions

Now that you have got this code to run, let's look at it more closely:

1. What are the intuitive meanings of `cons`, `car` and `cdr`?
2. What are the intuitive meanings of `h` and `h´`?
3. The operator `h´` is a homomorphism. What are its source and target algebras?

   Hint: what are the denotations of the modules defining the sort and arity of `h´`?
4. Run two reductions that show part (1) of the homomorphism definition (Definition 3.1 on page 20 of the course notes) holds for $h =$ `h`, and suitable choices for the operators $\sigma_M, \sigma_{M'}$ and for the terms a_i. Run another two reductions demonstrating that the equation also holds for a different choice of operators and terms.
5. Define and test an operator that finds the largest number in a list. Does this operation define a homomorphism?
6. You might have noticed that OBJ3 tells you how many rewrites it has performed. You might also have been surprised at how large that number is in some of these examples. To see what rules are applied, type

   ```
   set trace on .
   ```

   before running a reduction. Because there can be rather a lot of output, you may want to turn it off again, by typing

   ```
   set trace off .
   ```

   before running other reductions. Now explain why there are so many reductions.

7. Try

   ```
   red 0 .
   ```

   in the context of the module `HOMS`, and explain the result.

You should hand in copies of your log files, to show that you actually succeeded in running the code.