CafeOBJ Examples

```
-- FILE: /home/diacon/LANG/Cafe/prog/simple-nat.mod
-- CONTENTS: ADT specification of natural numbers
with addition and
--          multiplication
-- AUTHOR: Razvan Diaconescu
-- DIFFICULTY: *

mod! BARE-NAT {

  [ NzNat Zero < Nat ]

  op 0 : -> Zero
  op s_ : Nat -> NzNat
}

mod! SIMPLE-NAT {
  protecting(BARE-NAT)

  op _+_ : Nat Nat -> Nat {comm}

  eq N:Nat + s(M:Nat) = s(N + M) .
  eq N:Nat + 0 = N .
}
```

```
mod! TIMES-NAT {
  protecting(SIMPLE-NAT)

  op _*_ : Nat Nat -> Nat

  vars M N : Nat

  eq 0 * N = 0 .
  eq N * 0 = 0 .
  eq N * s(M) = (N * M) + N .
}

-- FILE: /home/diacon/LANG/Cafe/prog/nat-omega.mod
-- CONTENTS: ADT specification of natural numbers with
-- infinity
-- featuring extending importation mode and module sums
-- AUTHOR: Razvan Diaconescu
-- DIFFICULTY: **
```


-- AUTHOR: Razvan Diaconescu
-- DIFFICULTY: **

```
mod! NAT-OMEGA {
  extending(BARE-NAT)

  op omega :  -> Nat
  pred _<=_ : Nat Nat

  vars N M : Nat

  eq 0 <= s(N) = true .
  cq s(M) <= s(N) = true     if M <= N .

  eq s(omega) = omega .
  eq N <= omega = true .
}
```

```
Script started on Thu Feb 17 22:23:36 2000
fleck@tornado [101]% cafeobj^M^M
-- loading standard prelude^M
;; Loading file
/usr/pkg/cafeobj/1.4.2/cafeobj1.4/prelude/std.bin ...
;; Loading of file
/usr/pkg/cafeobj/1.4.2/cafeobj1.4/prelude/std.bin is
finished.

          -- CafeOBJ system Version 1.4.2 --
          built: 2000 Jan 19 Wed 16:41:02 GMT
               prelude file: std.bin
                     ***
             2000 Feb 18 Fri 4:23:44 GMT
                 Type ? for help
                     ---
                 built on CLISP
                     ---
                 built on CLISP
CafeOBJ> in simple-nat
processing input : ./simple-nat.mod
-- defining module! BARE-NAT..._* done.
-- defining module! SIMPLE-NAT.._..* done.
-- defining module! TIMES-NAT..._...* done.
```

```
CafeOBJ> select BARE-NAT
BARE-NAT> show sorts
* visible sorts :
  NzNat, NzNat < Nat
  Zero, Zero < Nat
  Nat, NzNat Zero < Nat
BARE-NAT> show ops
..............................(0)..............................
  * rank: -> Zero
..............................(s _)..............................
  * rank: Nat -> NzNat
..............................(Nat)..............................
  * rank: -> SortId
    - attributes:  { constr }
.............................(Zero).............................
  * rank: -> SortId
    - attributes:  { constr }
.............................(NzNat).............................
  * rank: -> SortId
    - attributes:  { constr }
BARE-NAT> show rules
 -- rewrite rules in module : BARE-NAT
```

```
BARE-NAT> show all rules
 -- rewrite rules in module : BARE-NAT
  1 : eq not true = false
  2 : eq not false = true
  3 : eq false and A:Bool = false
  4 : eq true or A:Bool = true
  5 : eq true and-also A:Bool = A:Bool
  6 : eq A:Bool and-also true = A:Bool
  7 : eq false and-also A:Bool = false
  8 : eq A:Bool and-also false = false
  9 : eq true or-else A:Bool = true
 10 : eq A:Bool or-else true = true
…

BARE-NAT> select SIMPLE-NAT
SIMPLE-NAT> show ops
.............................(_ + _).............................
  * rank: Nat Nat -> Nat^M

.............................(_ + _).............................
  * rank: Nat Nat -> Nat
    - attributes:  { comm }
    - axioms:
      eq N:Nat + s M:Nat = s (N:Nat + M:Nat)
      eq N:Nat + 0 = N:Nat
```

```
SIMPLE-NAT> show rules
 -- rewrite rules in module : SIMPLE-NAT
  1 : eq N:Nat + s M:Nat = s (N:Nat + M:Nat)
  2 : eq N:Nat + 0 = N:Nat
SIMPLE-NAT> red (s s 0) + (s s s 0) .
-- reduce in SIMPLE-NAT : s (s 0) + s (s (s 0))
s (s (s (s (s 0)))) : NzNat
(0.010 sec for parse, 6 rewrites(0.040 sec), 7 matches)
SIMPLE-NAT> red 0 + (n:Nat) .
-- reduce in SIMPLE-NAT : 0 + n:Nat
n:Nat : Nat
(0.000 sec for parse, 1 rewrites(0.000 sec), 2 matches)
SIMPLE-NAT> select TIMES-NAT
```

```
TIMES-NAT> show ops
..............................(_ * _)..............................
  * rank: Nat Nat -> Nat
    - axioms:
      eq 0 * N = 0
      eq N * 0 = 0
      eq N * s M = (N * M) + N
TIMES-NAT> red (s s 0) * (s s s 0) .
-- reduce in TIMES-NAT : s (s 0) * s (s (s 0))
s (s (s (s (s (s 0))))) : NzNat
(0.010 sec for parse, 19 rewrites(0.050 sec), 29 matches)
TIMES-NAT> in nat-omega
processing input : ./nat-omega.mod
-- defining module! NAT-OMEGA...._....* done.
TIMES-NAT> select NAT-OMEGA
```

```
NAT-OMEGA> show ops
............................(omega)............................
  * rank: -> Nat
............................(_ <= _)............................
  * rank: Nat Nat -> Bool
    - axioms:
  * rank: Nat Nat -> Bool
    - axioms:
      eq 0 <= s N = true
      ceq s M <= s N = true if M <= N
      eq N <= omega = true
NAT-OMEGA> red s s 0 <= s s s 0 .
-- reduce in NAT-OMEGA : s (s 0) <= s (s (s 0))
true : Bool
(0.010 sec for parse, 3 rewrites(0.030 sec), 10 matches)
NAT-OMEGA> red s s 0 <= 0 .
-- reduce in NAT-OMEGA : s (s 0) <= 0
s (s 0) <= 0 : Bool
(0.000 sec for parse, 0 rewrites(0.000 sec), 5 matches)
NAT-OMEGA> red s s 0 <= omega .
-- reduce in NAT-OMEGA : s (s 0) <= omega
true : Bool
(0.000 sec for parse, 1 rewrites(0.010 sec), 5 matches)
NAT-OMEGA> red omega <= s s 0 .
-- reduce in NAT-OMEGA : omega <= s (s 0)
omega <= s (s 0) : Bool
(0.010 sec for parse, 0 rewrites(0.000 sec), 5 matches)
NAT-OMEGA> open SIMPLE-NAT + NAT-OMEGA .
_*
-- opening module SIMPLE-NAT + NAT-OMEGA.. done.
%SIMPLE-NAT + NAT-OMEGA> eq omega + omega = omega .
%SIMPLE-NAT + NAT-OMEGA> red omega + (s s 0) .
*
```

```
-- reduce in %SIMPLE-NAT + NAT-OMEGA : omega + s (s 0)
omega : Nat
(0.010 sec for parse, 5 rewrites(0.040 sec), 11 matches)
%SIMPLE-NAT + NAT-OMEGA> set trace whole on
%SIMPLE-NAT + NAT-OMEGA> red s( (s 0) +  (s 0) .
-- reduce in %SIMPLE-NAT + NAT-OMEGA : s 0 + s 0
%SIMPLE-NAT + NAT-OMEGA> red (s 0) + (s 0) .
-- reduce in %SIMPLE-NAT + NAT-OMEGA : s 0 + s 0
[1]: s 0 + s 0
---> s (s 0 + 0)
[2]: s (s 0 + 0)
---> s (s (0 + 0))
[3]: s (s (0 + 0))
---> s (s 0)
s (s 0) : NzNat
(0.000 sec for parse, 3 rewrites(0.020 sec), 11 matches)
%SIMPLE-NAT + NAT-OMEGA> set trace off
%SIMPLE-NAT + NAT-OMEGA> set trace on
%SIMPLE-NAT + NAT-OMEGA> red omega + (s s 0).
-- reduce in %SIMPLE-NAT + NAT-OMEGA : omega + s (s 0)
1>[1] rule: eq N:Nat + s M:Nat = s (N:Nat + M:Nat)
   { N:Nat |-> omega, M:Nat |-> s 0 }
1<[1] omega + s (s 0) --> s (omega + s 0)
[1]: omega + s (s 0)
1<[1] omega + s (s 0) --> s (omega + s 0)
[1]: omega + s (s 0)
---> s (omega + s 0)
1>[2] rule: eq N:Nat + s M:Nat = s (N:Nat + M:Nat)
   { N:Nat |-> omega, M:Nat |-> 0 }
1<[2] omega + s 0 --> s (omega + 0)
[2]: s (omega + s 0)
---> s (s (omega + 0))
1>[3] rule: eq N:Nat + 0 = N:Nat
```

```
   { N:Nat |-> omega }
1<[3] omega + 0 --> omega
[3]: s (s (omega + 0))
---> s (s omega)
1>[4] rule: eq s omega = omega
   {}
1<[4] s omega --> omega
[4]: s (s omega)
---> s omega
1>[5] rule: eq s omega = omega
   {}
1<[5] s omega --> omega
[5]: s omega
---> omega
omega : Nat
(0.000 sec for parse, 5 rewrites(0.060 sec), 11 matches)
%SIMPLE-NAT + NAT-OMEGA> q
[Leaving CafeOBJ]
fleck@tornado [102]% exit
exit
```