Behavioral equations example

module* NAT-STREAM {
  protecting (SIMPLE-NAT)
  *[ Stream ]* -- hidden sort declaration
  bop __ : Nat Stream -> Stream -- binary op with no name
  bop hd : Nat Stream -> Nat
  bop tl : Nat Stream -> Stream
  op zeros: -> Stream
  var N : Nat
  var S : Stream
  eq hd (N S) = N .
  beq tl(N S) = S . -- tl(N S) and S are indistinguishable, not equal
  eq hd zeros = 0 .
  beq tl zeros = zeros . -- indistinguishable, not equal
}

From this tl(s(0) zeros) is not known to equal zeros, only behaviorally equal (i.e., indistinguishable). However, hd(tl(s(0) zeros)) is equal to hd(zeros).

Objects of a hidden sort are never “seen” by anyone. CafeOBJ rules require that operations on a hidden sort be declared “behavioral” (bop) and their properties be expressed as ‘beq’ (or bceq).

Matching under assoc/comm

CafeOBJ provides several “equational theory attributes”. Most useful are commutativity and associativity since both properties are frequently desirable but lead to non-terminating rewrite rules. When 'assoc' or 'comm' is declared for an operation, the system considers all appropriate rearrangements when searching for a substitution. For instance, in CafeOBJ '_and_' is associative.

  op _and_ : Bool Bool -> Bool { assoc }

so that a term

  true and false and true
is equal to

  (true and false) and true
and to

  true and (false and true).

So for example, the rewrite rule

  eq true and X:Bool = X .

does not directly apply to the term

  (true and false) and true,
but it none-the-less matches and yields

  false and true.
**Proving options**
The rewriting facilities can in fact be used to accomplish proofs of some (simple) assertions. For instance, in the SIMPLE-NAT module, there are only the equations

- \( eq \ 0 + N = N \).
- \( eq \ s(N) + M = s(N + M) \).

This only requires that 0 behave as we expect when used on the left (i.e., no 'comm' attribute). We can have the system perform the steps of an induction proof that 0 also behaves as we expect when used on the right.

```plaintext
open SIMPLE-NAT
SIMPLE-NAT > op a : -> Nat . -- 'a' is a new unrestricted constant of sort Nat
SIMPLE-NAT > reduce 0 + 0 . -- basis case
  0 : Zero -- 0 on right of 0 is OK
SIMPLE-NAT > eq a + 0 . -- induction hypothesis: assume 0 on right of 'a' is OK
SIMPLE-NAT > reduce s(a) + 0 .
  s(a) : NzNat -- induction extended — OK on next, proof complete

close
```