lib/basic/smt-fmaps.ath

```
# Module for rudimentary finite maps. This module is natively
# understood by the SMT translator, and it's how Athena handles
# SMT problems involving finite functions.

module FMap {

#datatype (FMap S T) := (empty-map T) | (update (FMap S T) S T)

datatype (FMap S T) := (empty-map T) | (update (Pair-Of S T) (FMap S T))

declare apply: (S, T) [(FMap S T) S] -> T

# assert* apply-axioms :=
#   (fun-def [(apply (empty-map ?default) _) --> ?default
#             (apply (update ?map ?key ?val) ?x) -->
#                [(?x = ?key) --> ?val
#                 _ --> (apply ?map ?x)]])

assert* apply-axioms :=
  [((apply (empty-map ?default) _) = ?default)
   (if (?x = ?key) ((apply (update (Pair ?key ?val) ?map) ?x) = ?val))
   (if (?x =/= ?key) ((apply (update (Pair ?key ?val) ?map) ?x) = (apply ?map ?x)))]

## Some testing:

define make-map :=
  lambda (L)
    match L {
       [] => (empty-map 0)
     | (list-of [x n] rest) => (update (Pair x n) (make-map rest))
    }

define update* :=
  lambda (fm pairs)
    letrec {loop := lambda (pairs res)
                      match pairs {
                        [] => res
                      | (list-of [key val] more) => (loop more (update (Pair  key val) res))}}
       (loop pairs fm)

define f := lambda (i) [(string->id ("s" joined-with (val->string i))) i]

define L := (from-to 1 5)

define sample-map := (make-map (map f L))

# So sample-map maps 's1 to 1, ..., 's5 to 5.

define applied-to := apply

(eval sample-map applied-to 's1)
(eval sample-map applied-to 's2)
(eval sample-map applied-to 's3)
(eval sample-map applied-to 's4)
(eval sample-map applied-to 's5)

# And this should give the default value 0:

(eval sample-map applied-to 's99)

}
```