lib/main/map_unittest.ath

```
load "map"

open Map

(define M1 [[1 --> 'a] [2 --> 'b] [1 --> 'c]])
(define M2 [['a --> true] ['b --> false] ['foo --> true]])
(eval M2 o M1)

(define [t1 t2] [(alist->map M2) (alist->map M1)])

define capitals :=
  [['paris --> 'france] ['tokyo --> 'japan] ['cairo --> 'egypt]]

define countries :=
  [['france --> 'europe] ['algeria --> 'africa] ['japan --> 'asia]]

(eval countries o capitals)

#(falsify composition-is-comm 20)

#(falsify composition-is-assoc 20)

#(falsify (close (((m3 o m2) o m1) = m3 o (m2 o m1))) 20)

#(falsify comp2-is-comm 10)

#(falsify comp2-is-assoc 80)

# (falsify comp2-app-lemma 10)

(eval [[1 --> 'a] [2 --> 'b]] <-> [[1 --> 'a] [3 --> 'c]])

(eval [[1 --> 'a] [2 --> 'b]] <-> [[1 --> 'a] [2 --> 'foo] [3 --> 'c]])

define compatible-theorem-1 := (forall m . m <-> m)

(falsify compatible-theorem-1 20)

define compatible-theorem-2 := (forall m1 m2 . m1 <-> m2 <==> m2 <-> m1)

(running-time (lambda () (falsify compatible-theorem-2 10)) 0)
# with new eval1: 4.22

define compatible-theorem-3 := (forall m1 m2 m3 . m1 <-> m2 & m2 <-> m3 ==> m1 <-> m3)

# (falsify compatible-theorem-3 10)

(define [s t hyp] [(apply ?tail:(Map 'T1 'T2)
                          ?k:'T1)

                  (apply ([?key:'T1 val] ++ ?tail:(Map 'T1 'T2))
                         ?k:'T1)

                  (?key:'T1 =/= ?k:'T1)])/

(assume hyp
  (!chain [s = t [apply-axioms]]))


define M' := [[1 --> 'a] [2 --> 'bar] [1 --> 'c]]

(eval agree-on M M' [1])
(eval agree-on M M' [2])

define ag-conjecture-1 :=
  (forall S m . agree-on m m S)

```

```
# Older map identity:
#assert* map-identity :=
#  (m1 = m2 <==> dom m1 = dom m2 & (agree-on m1 m2 dom m1));;

define agree-characterization :=
  (forall S m1 m2 . (agree-on m1 m2 S) <==> forall x . x in S ==> m1 applied-to x = m2 applied-to x)

# by-induction agree-characterization {
#   (S as Set.null) =>
#      pick-any m1 m2
#       let {dir1 := assume hyp := (agree-on m1 m2 null)
#                       pick-any x
#                         (!chain [(x in S) ==> false [Set.null-characterization]
#                                                ==> (m1 applied-to x = m2 applied-to x) [prop-taut]]);
#            dir2 := assume hyp := (forall x . x in null ==> m1 applied-to x = m2 applied-to x)
#                       (!chain-> [true ==> (agree-on m1 m2 S) [agree-on-axioms]])}
#         (!equiv dir1 dir2)
# | (S as (Set.insert h t)) =>
#      let {dir1 := assume hyp := (agree-on m1 m2 S)
# }

#(falsify agree-characterization 10)
#(!induction* agree-characterization)

(eval (empty-map:(Map Int Int) at  1))

(eval M applied-to 1)
(eval M applied-to 2)
(eval M applied-to 97)
(eval M - 1 applied-to 2)
(eval M - 1 applied-to 1)

(define M1 [[1 --> 'a] [2 --> 'b] [1 --> 'c]])

(define M2 [['a --> true] ['b --> false] ['foo --> true]])

(eval M2 o M1)

#(falsify composition-is-assoc 10)

define [n] := [?n:N]

let {m := (alist->map [[1 --> 2] [2 --> 3] [3 --> 1]]);
     _ := (print "\nm iterated once:   " (eval m ^^ 1));
     _ := (print "\nm iterated twice:  " (eval m ^^ 2));
     _ := (print "\nm iterated thrice: " (eval m ^^ 3))}
  (print "\nAre m and m^^3 identical?: " (eval m = m ^^ 3))

(eval [[1 --> 'a] [2 --> 'b]] <-> [[1 --> 'a] [3 --> 'c]])

(eval [[1 --> 'a] [2 --> 'b]] <-> [[1 --> 'a] [2 --> 'foo] [3 --> 'c]])

define compatible-theorem-1 := (forall m . m <-> m)

(falsify compatible-theorem-1 20)

define compatible-theorem-2 := (forall m1 m2 . m1 <-> m2 <==> m2 <-> m1)

#(running-time (lambda () (falsify compatible-theorem-2 50)) 0)
# with new eval1: 4.22

define compatible-theorem-3 := (forall m1 m2 m3 . m1 <-> m2 & m2 <-> m3 ==> m1 <-> m3)

#(falsify compatible-theorem-3 10)

#(define remove-correctness
#  (forall m x . (m - x) applied-to x = NONE))

#(holds? remove-correctness)
#(!induction* remove-correctness)
```

```
138
139 #(falsify remove-correctness 100)
140
141 #(define conj
142 #  (close (agree-on m1 m2 A) <==> m1 ^ A = m2 ^ A))
143 #?  (close (agree-on m1 m2 A) <==> m1 |^ A = m2 |^ A))
144
145 #(!induction* conj)
```