# lib/memory-range/memory_unittest.ath

```
## Test Memory-theory.

load "memory"

#########################################################
# Test proofs with standard operators:

define Mem-ops := no-renaming

assert (theory-axioms Memory.theory)

pick-any M:(Memory 'S) a:(Memory.Loc 'S) x:'S
         b:(Memory.Loc 'S)
  assume (a = b)
    (!chain [((M Memory.\ a Memory.<- x) Memory.at b) = x
             [(get-property Memory.assign.equal Mem-ops Memory.theory)]])

#(!map-method (method (n) (!property-test n Mem-ops Memory.theory))
#             (property-names Memory-theorems) make-conjunction)

open Memory

pick-any M:(Memory 'S) a:(Memory.Loc 'S) x:'S
         b:(Memory.Loc 'S)
  assume (a = b)
    (!chain [((M \ a <- x) at b) = x
             [(get-property Memory.assign.equal Mem-ops theory)]])

(!prove-property Double-assign Mem-ops theory)
(!prove-property Direct-double-assign Mem-ops theory)
(!prove-property Self-assign Mem-ops theory)
(!prove-property Direct-self-assign Mem-ops theory)
(!prove-property Double-swap Mem-ops theory)
(!prove-property Direct-double-swap Mem-ops theory)

#########################################################
# Test proofs with a different set of operators:

declare At: (S) [(Memory S) (Memory.Loc S)] -> S

declare Assign: (S) [(Memory.Loc S) S] -> (Memory.Change S)

declare Swap: (S) [(Memory.Loc S) (Memory.Loc S)] -> (Memory.Change S)

define Mem-ops := (renaming |{at := At, <- := Assign, swap := Swap}|)

assert (Mem-ops (theory-axioms theory))

(!prove-property Double-assign Mem-ops theory)
(!prove-property Direct-double-assign Mem-ops theory)
(!prove-property Self-assign Mem-ops theory)
(!prove-property Direct-self-assign Mem-ops theory)
(!prove-property Double-swap Mem-ops theory)
(!prove-property Direct-double-swap Mem-ops theory)
```