lib/memory-range/swap-implementation.ath

```
load "memory"

extend-module Memory {
  define t := ?t:(Loc 'S)

  define swap-open-implementation :=
    (forall M a b t M1 M2 M3 .
      a =/= t & b =/= t &
      M1 = M \ t <- (M at a) &
      M2 = M1 \ a <- (M1 at b) &
      M3 = M2 \ b <- (M2 at t)
      ==> M3 = (M \ t <- (M at a)) \ (swap a b))

  define swap-implementation :=
    (forall M a b x M1 M2 .
      x = (M at a) &
      M1 = M \ a <- (M at b) &
      M2 = M1 \ b <- x
      ==> M2 = M \ (swap a b))
#------------------------------------------------------------------------
define proofs :=
  method (theorem adapt)
    let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
         [at \ swap] := (adapt [at \ swap]);
         [eq uneq] := [assign.equal assign.unequal]}
     match theorem {
      (val-of swap-open-implementation) =>
      pick-any M:(Memory 'S) a:(Memory.Loc 'S) b:(Memory.Loc 'S)
               t:(Memory.Loc 'S) M1:(Memory 'S) M2:(Memory 'S)
               M3:(Memory 'S)
       let {i := (M1 = M \ t <- (M at a));
            ii := (M2 = M1 \ a <- (M1 at b));
            iii := (M3 = M2 \ b <- (M2 at t))}
        assume (a =/= t & b =/= t & i & ii & iii)
         conclude (M3 = (M \ t <- (M at a)) \ (swap a b))
          let {_ := (!sym (a =/= t));
               _ := (!sym (b =/= t));
               I := (!chain
                     [(M2 at t)
                    = ((M1 \ a <- (M1 at b)) at t)       [ii]
                    = (M1 at t)                          [uneq]
                    = ((M \ t <- (M at a)) at t)         [i]
                    = (M at a)                           [eq]]);
               II := (!chain
                      [(M3 at a)
                     = ((M2 \ b <- (M2 at t)) at a)      [iii]
                     = ((M2 \ b <- (M at a)) at a)       [I]]);
               III := conclude (M3 at a = M at b)
                       (!two-cases
                        assume (b = a)
                          (!chain
                           [(M3 at a)
                          = ((M2 \ b <- (M at a)) at a)  [II]
                          = (M at a)                     [eq]
                          = (M at b)                     [(b = a)]])
                         assume (b =/= a)
                           (!chain
                            [(M3 at a)
                           = ((M2 \ b <- (M at a)) at a) [II]
                           = (M2 at a)                   [uneq]
                           = ((M1 \ a <- (M1 at b)) at a)[ii]
                           = (M1 at b)                   [eq]
                           = ((M \ t <- (M at a)) at b)  [i]
                           = (M at b)                    [uneq]]));
               IV := pick-any u
                      conclude (M3 at u =
                                 ((M \ t <- (M at a)) \ (swap a b)) at u)
```

```
                       (!three-cases
                         assume (a = u)
                          (!combine-equations
                           (!chain
                            [(M3 at u)
                           = (M3 at a)                   [(a = u)]
                           = (M at b)                    [III]
                           = ((M \ t <- (M at a)) at b)  [uneq]])
                           (!chain
                            [(((M \ t <- (M at a)) \ (swap a b)) at u)
                           = (((M \ t <- (M at a)) \ (swap a b)) at a)
                                                         [(a = u)]
                           = ((M \ t <- (M at a)) at b)  [swap.equal1]]))
                         assume (b = u)
                          (!combine-equations
                           (!chain
                            [(M3 at u)
                           = (M3 at b)                    [(b = u)]
                           = ((M2 \ b <- (M2 at t)) at b) [iii]
                           = (M2 at t)                    [eq]
                           = (M at a)                     [I]
                           = ((M \ t <- (M at a)) at a)   [uneq]])
                           (!chain
                            [(((M \ t <- (M at a)) \ (swap a b)) at u)
                             = (((M \ t <- (M at a)) \ (swap a b)) at b)
                                                          [(b = u)]
                             = ((M \ t <- (M at a)) at a) [swap.equal2]]))
                         assume (a =/= u & b =/= u)
                          (!combine-equations
                           (!chain
                            [(M3 at u)
                           = ((M2 \ b <- (M2 at t)) at u) [iii]
                           = (M2 at u)                    [uneq]
                           = ((M1 \ a <- (M1 at b)) at u) [ii]
                           = (M1 at u)                    [uneq]
                           = ((M \ t <- (M at a)) at u)   [i]])
                           (!chain
                            [(((M \ t <- (M at a)) \ (swap a b)) at u)
                             = ((M \ t <- (M at a)) at u) [swap.unequal]]))))}
               (!chain
                [M3 = ((M \ t <- (M at a)) \ (swap a b))  [equality]])
    | (val-of swap-implementation) =>
      pick-any M:(Memory 'S) a:(Memory.Loc 'S) b:(Memory.Loc 'S) x:'S
               M1:(Memory 'S) M2:(Memory 'S)
       let {i := (x = (M at a));
            ii := (M1 = M \ a <- (M at b));
            iii := (M2 = M1 \ b <- x)}
       assume (i & ii & iii)
         conclude (M2 = M \ (swap a b))
          let {I := (!chain
                      [(M2 at a)
                     = ((M1 \ b <- x) at a)             [iii]
                     = ((M1 \ b <- (M at a)) at a)      [i]]);
               II := conclude (M2 at a = M at b)
                       (!two-cases
                        assume (b = a)
                          (!chain
                           [(M2 at a)
                          = ((M1 \ b <- (M at a)) at a) [I]
                          = (M at a)                    [eq]
                          = (M at b)                    [(b = a)]])
                        assume (b =/= a)
                          (!chain
                           [(M2 at a)
                          = ((M1 \ b <- (M at a)) at a) [I]
                          = (M1 at a)                   [uneq]
                          = ((M \ a <- (M at b)) at a)  [ii]
                          = (M at b)   [eq]]));
               III :=
                pick-any u
```

```
                 conclude (M2 at u = (M \ (swap a b)) at u)
                  (!three-cases
                   assume (a = u)
                     (!combine-equations
                      (!chain
                       [(M2 at u)
                      = (M2 at a)                       [(a = u)]
                      = (M at b)                        [II]])
                      (!chain
                       [((M \ (swap a b)) at u)
                      = ((M \ (swap a b)) at a)         [(a = u)]
                      = (M at b)                        [swap.equal1]]))
                   assume (b = u)
                     (!combine-equations
                      (!chain
                       [(M2 at u)
                      = (M2 at b)                       [(b = u)]
                      = ((M1 \ b <- x) at b)            [iii]
                      = x                               [eq]
                      = (M at a)                        [i]])
                      (!chain
                       [((M \ (swap a b)) at u)
                      = ((M \ (swap a b)) at b)         [(b = u)]
                      = (M at a)                        [swap.equal2]]))
                   assume (a =/= u & b =/= u)
                    (!combine-equations
                     (!chain
                      [(M2 at u)
                     = ((M1 \ b <- x) at u)             [iii]
                     = (M1 at u)                        [uneq]
                     = ((M \ a <- (M at b)) at u)       [ii]
                     = (M at u)                         [uneq]])
                     (!chain
                      [((M \ (swap a b)) at u)
                     = (M at u)                         [swap.unequal]]))))}
          (!chain [M2 = (M \ (swap a b))                [equality]])
    }

(add-theorems theory
              |{[swap-open-implementation swap-implementation] := proofs}|)
} # Memory
```