lib/memory-range/temp.ath

```
define proof :=
  method (theorem adapt)
    let {[get prove chain chain-> chain<-] := (proof-tools adapt Theory);
         deref := (adapt deref)}
    match theorem {
      (val-of correctness) =>
      by-induction (adapt theorem) {
        (stop q:(It 'X 'S)) =>
        pick-any i:(It 'X 'S) j:(It 'X 'S)
                 M:(Memory 'S) k:(It 'Y 'S)
                 M':(Memory 'S) k':(It 'Y 'S)
          assume A := ((limits i j) = SOME stop q &
                       ~ k *in stop q &
                       M' = M \ (copy-memory i j k) &
                       k' = M \\ (copy i j k))
            conclude goal :=
                (exists ?r' .
                  (limits k k') = SOME ?r' &
                  (collect M' ?r') = (collect M stop q) &
                  forall ?h . ~ ?h *in ?r' ==>
                              M' at deref ?h = M at deref ?h)
              let {EL1 := (!prove empty-limits1);
                   _ := conclude (i = j)
                          (!chain-> [(limits i j)
                                     = (SOME stop q)   [A]
                                     ==> (i = j)       [EL1]]);
                   _ := conclude (M' = M)
                          (!chain->
                           [M' = (M \ (copy-memory i j k)) [A]
                               = M              [copy-memory.empty]]);
                   _ := conclude (k' = k)
                         (!chain->
                          [k' = (M \\ (copy i j k))   [A]
                              = k                    [empty]]);
                   _ := (!chain [(start stop k)
                                 = k                  [start.of-stop]
                                 = (finish stop k) [finish.of-stop]]);
                   protected := pick-any h
                                  assume (~ h *in stop k)
                                    (!chain
                                     [(M' at deref h)
                                      = (M at deref h)  [(M' = M)]]);
                   EL := (!prove empty-limits);
                   B := (!both
                         (!chain [(limits k k')
                                  = (limits k k)    [(k' = k)]
                                  = (SOME stop k)   [EL]])
                         (!both (!chain
                                 [(collect M' stop k)
                                = nil:(List 'S)     [collect.of-stop]
                                = (collect M stop q)[collect.of-stop]])
                                protected))}
              (!chain-> [B ==> goal [existence]])
        | (back r:(Range 'X 'S)) =>
           let {ind-hyp :=
                (forall ?i:(It 'X 'S) ?j:(It 'X 'S)
                        ?M:(Memory 'S) ?k:(It 'Y 'S)
                        ?M':(Memory 'S) ?k':(It 'Y 'S) .
                 (limits ?i ?j) = SOME r &
                 ~ ?k *in r &
                 ?M' = ?M \ (copy-memory ?i ?j ?k) &
                 ?k' = ?M \\ (copy ?i ?j ?k)
                 ==> exists ?r' .
                       (limits ?k ?k') = SOME ?r' &
                       (collect ?M' ?r') = (collect ?M r) &
                       forall ?h . ~ ?h *in ?r' ==>
                                   ?M' at deref ?h = ?M at deref ?h)}
```

```
            pick-any i:(It 'X 'S) j:(It 'X 'S) M:(Memory 'S) k:(It 'Y 'S)
                     M':(Memory 'S) k':(It 'Y 'S)
              let {M1 := (M \ deref k <- M at deref i);
                   A1 := ((limits i j) = SOME back r);
                   A2 := (~ k *in back r);
                   A3 := (M' = M \ (copy-memory i j k));
                   A4 := (k' = M \\ (copy i j k))}
              assume (A1 & A2 & A3 & A4)
                conclude
                  goal := (exists ?r' .
                           (limits k k') = SOME ?r' &
                           (collect M' ?r') = (collect M back r) &
                           forall ?h . ~ ?h *in ?r' ==>
                                       M' at deref ?h = M at deref ?h)
                  let {(and B1 B2) :=
                         (!chain->
                          [(limits i j)
                           = (SOME back r) [A1]
                           = (limits (start back r) (finish back r))
                                                 [limits.collapse]
                           ==> (i = (start back r) & j = (finish back r))
                                                 [limits.injective]]);
                       _ := (!chain->
                              [true
                               ==> ((start back r) =/= (finish back r))
                                          [finish.nonempty-back]
                               ==> (i =/= j)  [B1 B2]]);
                       RR := (!prove *in.range-reduce);
                       CU := (!prove collect.unchanged);
                       B3 := (!chain-> [A2 ==> (~ k *in r)      [RR]
                                           ==> ((collect M1 r) =
                                                (collect M r)) [CU]]);
                       B4 := conclude (M' = (M1 \ (copy-memory
                                                    (successor i) j
                                                    (successor k))))
                               (!chain
                                [M' = (M \ (copy-memory i j k)) [A3]
                                    = (M1 \ (copy-memory (successor i) j
                                                         (successor k)))
                                                [copy-memory.nonempty]]);
                       B5 := conclude (k' = (M1 \\ (copy (successor i) j
                                                        (successor k))))
                              (!chain
                               [k' = (M \\ (copy i j k)) [A4]
                                   = (M1 \\ (copy (successor i) j
                                                  (successor k)))
                                                        [nonempty]]);
                       LB := (!prove limits-back);
                       A1' := (!chain->
                               [A1 ==> ((limits (successor i) j) =
                                        SOME r)              [LB]]);
                       RS2 := (!prove *in.range-shift2);
                       B6 := (!chain->
                              [A2
                           ==> (~ (successor k) *in r)    [RS2]
                           ==> (A1' & ~ (successor k) *in r & B4 & B5)
                                                          [augment]
                           ==> (exists ?r' .
                                 (limits (successor k) k') = SOME ?r' &
                                 (collect M' ?r') = (collect M1 r) &
                                 forall ?h . ~ ?h *in ?r' ==>
                                             M' at deref ?h =
                                             M1 at deref ?h) [ind-hyp]])}
                  pick-witness r'' for B6 B6-w
                    (!force goal)
        } # by-induction
      } # match theorem

(evolve Theory [[correctness] proof])
} # copy
```

```
} # Forward-Iterator
```