#..........................................................................
load "forward-iterator"

#..........................................................................
extend-module Forward-Iterator {

declare replace: (S, X) \{\{It X S\} \{It X S\} S S\} -> (Memory.Change S)

module replace {

define axioms :=

  \{(fun [M \ (replace i j x y)] =
   [M when (i = j)
    (M \ (deref i) <- y) \ (replace (successor i) j x y))
   when (i /= j & M at deref i = x)
    (M \ (replace (successor i) j x y))
    when (i /= j & M at deref i /= x)]\}

define [if-empty if-equal if-unequal] := axioms

(add-axioms theory axioms)

define replace' := List.replace

define \(M'\) := ?M'; (Memory \'S\)

define q := ?q:(It \'Z \'(S)

define correctness := (forall r . correctness-prop r)

define proof :=

  method (theorem adapt)
    let \{\{get prove chain chain-> chain<-\} := (proof-tools adapt theory);
    [deref \in successor] := (adapt [deref \in successor])

    match theorem {
      (val-of correctness) =>
      by-induction (adapt theorem) {
        (stop h:(It \'X \'(S)) =>
          pick-any M:(Memory \'S) M':(Memory \'S) i:(It \'X \'(S)
            x:'S y:'S
          j:(It \'X \'(S)
            x:'S y:'S
            (collect M stop h) = (replace' (collect M stop h) x y))
          assume (A1 & A2)

          let \(ER1 := (!prove empty-range1);
            _ := conclude (i = j)
            (!chain-> [A1 ==\( (i = j) [ER1]\)]);
          _ := conclude (M' = M)

            (!chain
              [M' = (M \ (replace i j x y))] [A2]
              = M
              [(i = j) if-empty]);
        B1 := conclude ((collect M stop h) =
          (replace' (collect M stop h) x y))

          (!chain
            [(collect M stop h)
              = (collect M stop h) [\(M' = M\)]
              = nil:(List \'S)
              [collect.of-stop]
              = (replace' nil x y) [List.replace.empty]
              = (replace' (collect M stop h) x y)
              [collect.of-stop]);
        B2 := conclude
      }
    }

  method (theorem adapt)
    let \{\{get prove chain chain-> chain<-\} := (proof-tools adapt theory);
    [deref \in successor] := (adapt [deref \in successor])

    match theorem {
      (val-of correctness) =>
      by-induction (adapt theorem) {
        (stop h:(It \'X \'(S)) =>
          pick-any M:(Memory \'S) M':(Memory \'S) i:(It \'X \'(S)
            x:'S y:'S
          j:(It \'X \'(S)
            x:'S y:'S
            (collect M stop h) = (replace' (collect M stop h) x y))
          assume (A1 & A2)

          let \(ER1 := (!prove empty-range1);
            _ := conclude (i = j)
            (!chain-> [A1 ==\( (i = j) [ER1]\)]);
          _ := conclude (M' = M)

            (!chain
              [M' = (M \ (replace i j x y))] [A2]
              = M
              [(i = j) if-empty]);
        B1 := conclude ((collect M stop h) =
          (replace' (collect M stop h) x y))

          (!chain
            [(collect M stop h)
              = (collect M stop h) [\(M' = M\)]
              = nil:(List \'S)
              [collect.of-stop]
              = (replace' nil x y) [List.replace.empty]
              = (replace' (collect M stop h) x y)
              [collect.of-stop]);
        B2 := conclude
      }
    }

}

lib/memory-range/replace-range.ath
\{(forall \ ?k:\ (It \ 'S') \ . \ ~ \ ?k \ *in \ stop \ h ==\>
M' \ at \ deref \ ?k = M \ at \ deref \ ?k)\}

\textbf{pick-any} \ k:\ (It \ 'S')
\begin{align*}
&\text{assume} \ (\sim \ k \ *in \ stop \ h) \\
&\text{\{\!chain \ \{(M' \ at \ deref \ k) = (M \ at \ deref \ k)\} \}}
\end{align*}

\{(\!both \ B1 \ B2)\}
\begin{align*}
\text{\{\r as \ \{\text{back \ r}:(\text{Range \ 'X')}) \ \} ==} \\
\text{\{\text{correctness-prop \ r}'\}}
\end{align*}

\textbf{pick-any} \ M:\ (Memory \ 'S') \ M':(Memory \ 'S') \ i:(It \ 'X') \ j:(It \ 'X') \ x: \ 'S' \ y: \ 'S'
\begin{align*}
\text{\{\text{\!chain} \ \{(M \ at \ deref \ i) = (M' \ at \ deref \ i)\} \}}
\end{align*}
\begin{align*}
\text{\{\text{\!both} \ B1 \ B2\}}
\end{align*}
\begin{align*}
&\text{\{\text{\!combine-equations} \ \{\text{\!chain} \ \{(collect \ M \ r) = (collect \ M' \ r)\} \ \}}} \\
&\text{\{\text{\!chain} \ \{(M \ at \ deref \ i) = (M' \ at \ deref \ i)\}) \ \}}
\end{align*}


= (y :: (replace' (collect M1 r') x y))

[assign.equal]

= (y :: (replace' (collect M r') x y))

[CU])

(!chain
 [(replace' (collect M r) x y)
  = (replace' ((M at deref i) ::
    (collect M r'))
   x y) [B4]
  collect.of-back]

= (y :: (replace' (collect M r') x y))

[List.replace.equal])

_ := conclude B2

pick-any h

assume D := (~ h *in r)

let (E :=

  (!chain->
   [D ==> (deref h = deref start r | h *in r')] [*in.of-back]
   ==> (~ (deref h = deref i | h *in r')) [B4]
   ==> (deref h /= deref i & ~ h *in r') [dm]
   ==> (deref h /= deref i)
   ==> (deref h /= deref h)
   [sym])})

(!chain->
 [D ==> (~ h *in r') [RR]
 ==> (M' at deref h = M1 at deref h) [C2b]
 ==> (M' at deref h = M at deref h) [E assign.unequal]])

(!both B1 B2)

assume (M at deref i /= x)

let (M1 := M;

C1 := (!chain
 [M' = (M \ (replace i j x y)) [A2]
 = (M \ (replace
 (successor i) j x y))
 [if-unequal]])

(and C2a C2b) :=

(!chain->
 [C1 ==> (B3 & C1) [augment]
 ==> ((collect M' r') = (replace' (collect M r') x y) &
 forall h . ~ h *in r' ==> M' at deref h = M at deref h) [ind-hyp]])

C3 := (!chain->
 [true ==> (~ start r *in r') [FNIR]
 ==> (~ i *in r') [B4]])

_ := {!sym (M at deref i /= x))}

_ := conclude B1

(!combine-equations
 (!chain
 [(collect' (collect M r) x y)
  = ((M' at deref i) ::
    (collect M' r')) [B4]
  collect.of-back]
  = ((M at deref i) ::
    (replace' (collect M r') x y))
  [C2a C2b]])

(!chain
 [(replace' (collect M r) x y)
  = (replace' ((M at deref i) ::
    (collect M r')) x y)
let memory-range/replace-range.ath

[B4
  collect.of-back]

  = ((M at deref i) ::
    (replace' (collect M r') x y))

  [List.replace.unequal]]));

_ := conclude B2

pick-any h

assume D := (¬ h *in r) (!chain->
  [D =>> (¬ h *in r') [RR]
  ==> (M' at deref h = M at deref h)
  [C2b])])

  (!both B1 B2))

  ) by-induction

) # by-induction

) # replace

) # Forward-Iterator