lib/memory-range/bidirectional-iterator.ath

load "forward-iterator"

module Bidirectional-Iterator {
  open Forward-Iterator

  declare predecessor: (X, S) [(It X S)] -> (It X S)

  module predecessor {
    define of-start :=
      (forall r . predecessor start r = start back r)
    define of-successor :=
      (forall i . predecessor successor i = i)
  }

  define theory :=
    (make-theory [Forward-Iterator]
      [predecessor.of-start predecessor.of-successor])

  define successor-of-predecessor :=
    (forall i . successor predecessor i = i)

  define proof :=
    method (theorem adapt)
      let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
        [successor predecessor] := (adapt [successor predecessor])}
      match theorem {
        (val-of successor-of-predecessor) =>
          pick-any i:(It 'X 'S)
          ![chain
            [(successor predecessor i)
              = (successor predecessor start stop i) [start.of-stop]
              = (successor start back stop i) [predecessor.of-start]
              = (start stop i) [successor.of-start]
              = i [start.of-stop]]}
      }
      (add-theorems theory |{[successor-of-predecessor] := proof}|)

} # Bidirectional-Iterator