Double Linked Lists : Desired Operations

- Define an ASM which offers the following operations, predicates and functions on double linked lists, whose elements have values in a given set VALUE:
  - CreateList (VALUE) : create a new double linked list with elements taking values in Value
  - Append (L, Val) : append at the end a new element with given value
  - Insert (L, Val, Elem) : insert after Elem in L a new element with Val
  - Delete (L, Elem) : delete Elem from L
  - AccessByValue (L, Val) : return the first element in L with Val
  - AccessByIndex (L, i) : return the i-th element in L
  - empty (L), length (L), occurs (L, Elem), position (L, Elem)
  - Update (L, Elem, Val) : update the the value of Elem in L to Val
  - Cat (L1,L2) : concatenate two given lists in the given order
  - Split (L, Elem, L1, L2) : split L into L1, containing L up to including Elem, and L2 containing the rest list of L

Double Linked Lists : Desired Properties

- Prove that the Linked List ASM has the following properties:
  - If the next-link of a list element Elem points to Elem', then the previous-link of Elem' points to Elem.
  - L is empty iff the next-link of its head points to its tail.
  - The set ELEM (L) of elements occurring in a list is the set of all E which can be reached, starting from the list head, by applying next-links until the list tail is encountered.
  - After applying Append (L, Val), the list is not empty.
  - A newly created linked list is empty and its length is 0.
  - By Append/Delete the list length in/de-creases by 1.
  - For non empty L and arbitrary elements E the following holds:

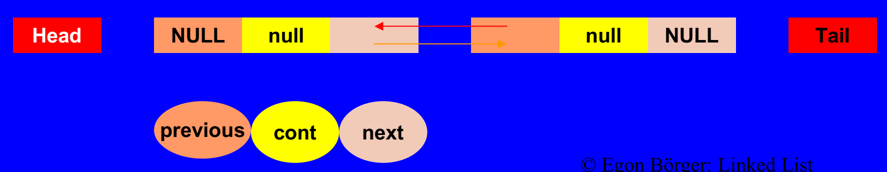
    Append (Delete (L,E),E) = Delete (Append (L,E),E)

A Problem Solution

- E. Börger, R. Stärk: Abstract State Machines. A Method for High-Level System Design and Analysis Springer-Verlag 2003, see http://www.di.unipi.it/AsmBook
  - See exercise in Chapter 2

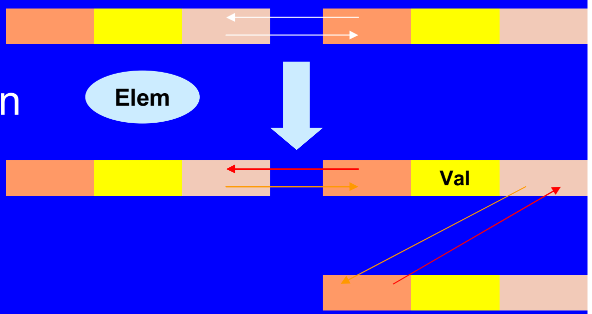
Double Linked Lists : Signature

- LINKED-LIST (VALUE) : dynamic set, with fcts "pointing" to structures of the following form (often VALUE suppressed) :
  - dynamic set ELEM (L) of "objects" curryly listed in L
  - distinguished elems Head (L), Tail (L) ∈ ELEM (L)
  - previous (L), next (L): ELEM (L) → ELEM (L) dyn link fcts
  - cont (L) : ELEM (L) → VALUE yields curr value of list elems
- initialize(L) for L ∈ LINKED-LIST (as usual, L is suppressed) as empty linked list with values in VALUE, defined as follows:
  - ELEM := { Head, Tail } next (Head) := Tail previous (Tail) := Head
  - previous (Head) := next (Tail) := null (ELEM) Head/Tail start/end the list
  - cont (Head) := cont (Tail) := null (VALUE) Head/Tail have no content



Double Linked Lists : Definition of Operations (1)

- CreateList (VALUE) ≡
  let L = new (LINKED-LIST (VALUE)) in initialize (L)
- Append (L, Val) ≡
  let e = new (ELEM (L)) in
  Link previous (Tail) & e
  Link e &Tail
  cont (e) := Val



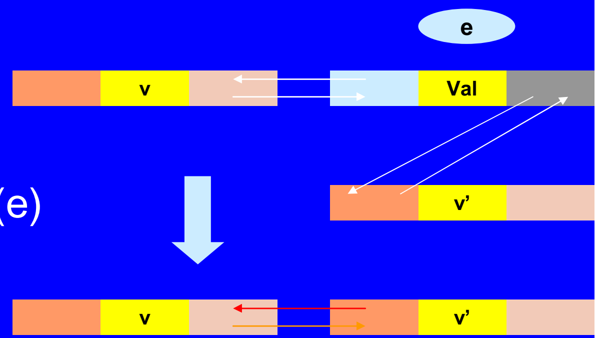
- Insert (L, Val, Elem) ≡ let e = new **(ELEM (L))** in
  cont (e) := Val
  Link Elem & e
  Link e & next (Elem)

  with Link a&b ≡ next (a) := b
  previous (b) := a

Double Linked Lists : Operations & Derived Fcts (2)

Delete (L, e) ≡

Link previous (e) & next (e)



length (L) ≡ ι m (next $^{m+1}$ (Head) = Tail) well defined by initialization

occurs (L, e) ≡ ∃ i ≤ length (L) : next ^{i} (Head) = e (e ∈ ELEM(L))

position (L, Elem) ≡ ιm (next ^{m} (Head) = Elem) if occurs (L, Elem)

AccessByIndex (L, i) ≡ next ^{i} (Head) if i ≤ length (L)

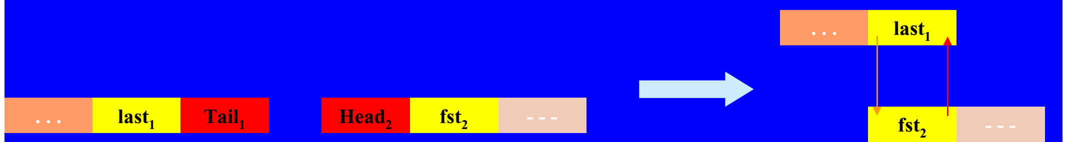
AccessByValue (L, Val) ≡ next ^{m} (Head) fst occ of Val

where m= min { i | cont (next ^{i} (Head)) = Val } is defined

Double Linked Lists : Definition of Operations (3)

Update (L, Elem, Val) ≡ If occurs (L, Elem)
then cont (Elem) := Val
else error msg "Elem does not occur in L"

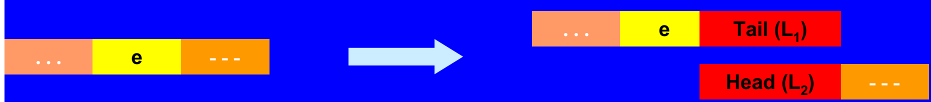
Cat (L_1, L_2) ≡ let L = new (LINKED-LIST) in
Head (L) := Head (L_1)
Tail (L) := Tail (L_2)
Link (L) previous (L_1) (Tail (L_1)) & next (L_2) (Head (L_2))
forall e∈ELEM (L) - {previous (L_1) (Tail (L_1)),Tail (L_1) }
Link (L) e & next (L_1) (e)
forall e∈ELEM (L_2) - { Head (L_2) ,Tail (L_2) }
Link (L) e & next (L_2) (e)



Double Linked Lists : Definition of Operations (4)

Split (L, e, L_1, L_2) ≡ let e_1 = new-tail, e_2 = new-head
Head (L_1) := Head (L)
Tail (L_1) := e_1
Link (L_1) e & e_1
forall E∈ELEM (L) if position (L, E) < position (L, e)
then Link (L_1) E & next (L) (E)
Head (L_2) := e_2
Link (L_2) e_2 & next (L) (e)
Tail (L_2) := Tail (L)
forall E∈ELEM (L) if position (L, e) < position (L, E)
then Link (L_2) E & next (L) (E)

where **e' = new-tail/head** ≡
cont (e') **:= null (VALUE)**
next/previous (e') **:= null (ELEM)**



Double Linked Lists : Proving the Properties (1)

- If the next-link of a list element Elem points to Elem', then the previous-link of Elem' points to Elem.
  - Initially true by defn of initialize (L), preserved by each opn due to the defn of Link (L) and the fact that next/previous are modified only using this macro.
- L is empty iff
  the next-link of its head points to its tail.
- A newly created linked list is empty and
  its length is 0.
- After applying Append (L, Val), the list is not empty
- By Append/Delete the list length in/de-creases
  by 1.

Double Linked Lists : Proving the Properties (2)

- For L≠[ ]: Append (Delete (L,E),E) = Delete (Append (L,E),E)
  - Follow from the defn of initialize (L), length (L), Append, Delete & the fact that Append/Insert yield a non null cont.
- The set ELEM (L) of elements occurring in a list is the set of all E which can be reached, starting from the list head, by applying next-links until the list tail is encountered.
  - Follows from the defn of ELEM(L).