

CLASS NOTES FOR CS 818A3 - SPRING 2005

# AN INTRODUCTION TO SEPARATION LOGIC

## 4. Lists and List Segments

John C. Reynolds  
Department of Computer Science  
Carnegie Mellon University

February 11, 2005

©2005 John C. Reynolds

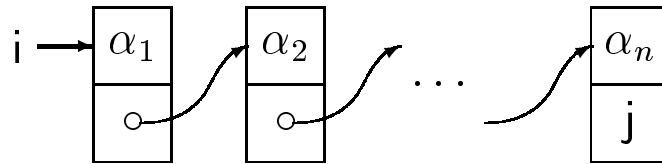
## Notation for Sequences

When  $\alpha$  and  $\beta$  are sequences, we write

- $\epsilon$  for the empty sequence.
- $[x]$  for the single-element sequence containing  $x$ . (We will omit the brackets when  $x$  is not a sequence.)
- $\alpha \cdot \beta$  for the composition of  $\alpha$  followed by  $\beta$ .
- $\alpha^\dagger$  for the reflection of  $\alpha$ .
- $\#\alpha$  for the length of  $\alpha$ .
- $\alpha_i$  for the  $i$ th component of  $\alpha$ .

## Singly-linked List Segments

$\text{lseg } \alpha (i, j)$ :



is defined by induction on the length of the sequence  $\alpha$  (i.e., by structural induction on  $\alpha$ ):

$$\text{lseg } \epsilon (i, j) \stackrel{\text{def}}{=} \mathbf{emp} \wedge i = j$$

$$\text{lseg } a \cdot \alpha (i, k) \stackrel{\text{def}}{=} \exists j. i \mapsto a, j * \text{lseg } \alpha (j, k).$$

## Properties

$$\text{lseg } a (i, j) \Leftrightarrow i \mapsto a, j$$

$$\text{lseg } \alpha \cdot \beta (i, k) \Leftrightarrow \exists j. \text{lseg } \alpha (i, j) * \text{lseg } \beta (j, k)$$

$$\text{lseg } \alpha \cdot b (i, k) \Leftrightarrow \exists j. \text{lseg } \alpha (i, j) * j \mapsto b, k$$

$$\text{list } \alpha i \Leftrightarrow \text{lseg } \alpha (i, \mathbf{nil}).$$

## Proof of the Composition Property

$$\text{lseg } \alpha \cdot \beta (i, k) \Leftrightarrow \exists j. \text{lseg } \alpha (i, j) * \text{lseg } \beta (j, k)$$

The proof is by induction on the length of  $\alpha$ .

When  $\alpha$  is empty:

$$\begin{aligned} & \exists j. \text{lseg } \epsilon (i, j) * \text{lseg } \beta (j, k) \\ & \Leftrightarrow \exists j. (\mathbf{emp} \wedge i = j) * \text{lseg } \beta (j, k) \\ & \Leftrightarrow \exists j. (\mathbf{emp} * \text{lseg } \beta (j, k)) \wedge i = j \\ & \Leftrightarrow \exists j. \text{lseg } \beta (j, k) \wedge i = j \\ & \Leftrightarrow \text{lseg } \beta (i, k) \\ & \Leftrightarrow \text{lseg } \epsilon \cdot \beta (i, k) \end{aligned}$$

When  $\alpha = a \cdot \alpha'$ :

$$\begin{aligned} & \exists j. \text{lseg } a \cdot \alpha' (i, j) * \text{lseg } \beta (j, k) \\ & \Leftrightarrow \exists j, l. i \mapsto a, l * \text{lseg } \alpha' (l, j) * \text{lseg } \beta (j, k) \\ & \Leftrightarrow \exists l. i \mapsto a, l * \text{lseg } \alpha' \cdot \beta (l, k) \quad (\text{induction hypothesis}) \\ & \Leftrightarrow \text{lseg } a \cdot \alpha' \cdot \beta (i, k) \end{aligned}$$

## Emptiness Conditions

$$\text{lseg } \alpha (i, j) \Rightarrow (i = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge j = \mathbf{nil}))$$

$$\text{lseg } \alpha (i, j) \Rightarrow (i \neq j \Rightarrow \alpha \neq \epsilon).$$

But these formulas do not say whether  $\alpha$  is empty when  $i = j \neq \mathbf{nil}$ .

## Nontouching List Segments

When

$$\text{lseg } a_1 \cdots a_n (i_0, i_n),$$

we have

$$\exists i_1, \dots, i_{n-1}.$$

$$(i_0 \mapsto a_1, i_1) * (i_1 \mapsto a_2, i_2) * \cdots * (i_{n-1} \mapsto a_n, i_n).$$

Thus  $i_0, \dots, i_{n-1}$  are distinct, but  $i_n$  is not constrained, and may equal any of the  $i_0, \dots, i_{n-1}$ . In this case, we say that the list segment is *touching*.

We can define nontouching list segments inductively by:

$$\text{ntlseg } \epsilon (i, j) \stackrel{\text{def}}{=} \mathbf{emp} \wedge i = j$$

$$\text{ntlseg } a \cdot \alpha (i, k) \stackrel{\text{def}}{=} i \neq k \wedge i \neq k + 1 \wedge (\exists j. i \mapsto a, j * \text{ntlseg } \alpha (j, k)),$$

or equivalently, we can define them in terms of  $\text{lseg}$ :

$$\text{ntlseg } \alpha (i, j) \stackrel{\text{def}}{=} \text{lseg } \alpha (i, j) \wedge \neg j \hookrightarrow -.$$

The obvious advantage of knowing that a list segment is nontouching is that it is easy to test whether it is empty:

$$\text{ntlseg } \alpha (i, j) \Rightarrow (\alpha = \epsilon \Leftrightarrow i = j).$$

Fortunately, there are common situations where list segments must be nontouching:

$$\text{list } \alpha \ i \Rightarrow \text{ntlseg } \alpha (i, \mathbf{nil})$$

$$\text{lseg } \alpha (i, j) * \text{list } \beta \ j \Rightarrow \text{ntlseg } \alpha (i, j) * \text{list } \beta \ j$$

$$\text{lseg } \alpha (i, j) * j \hookrightarrow - \Rightarrow \text{ntlseg } \alpha (i, j) * j \hookrightarrow -.$$

## Inserting a List Element

- At the beginning:

$$\begin{aligned} & \{\text{lseg } \alpha (i, j)\} \\ & k := \mathbf{cons}(a, i); \\ & \{k \mapsto a, i * \text{lseg } \alpha (i, j)\} \\ & \{\exists i. k \mapsto a, i * \text{lseg } \alpha (i, j)\} \\ & \{\text{lseg } a \cdot \alpha (k, j)\} \\ & i := k \\ & \{\text{lseg } a \cdot \alpha (i, j)\} \end{aligned}$$

- At the end of a nonempty segment:

$$\begin{aligned} & \{\text{lseg } \alpha (i, j) * j \mapsto a, k\} \\ & l := \mathbf{cons}(b, k); \\ & \{\text{lseg } \alpha (i, j) * j \mapsto a, k * l \mapsto b, k\} \\ & \{\text{lseg } \alpha (i, j) * j \mapsto a * j + 1 \mapsto k * l \mapsto b, k\} \\ & \{\text{lseg } \alpha (i, j) * j \mapsto a * j + 1 \mapsto - * l \mapsto b, k\} \\ & [j + 1] := l \\ & \{\text{lseg } \alpha (i, j) * j \mapsto a * j + 1 \mapsto l * l \mapsto b, k\} \\ & \{\text{lseg } \alpha (i, j) * j \mapsto a, l * l \mapsto b, k\} \\ & \{\text{lseg } \alpha \cdot a (i, l) * l \mapsto b, k\} \\ & \{\text{lseg } \alpha \cdot a \cdot b (i, k)\} \end{aligned}$$

## Deleting a List Element

- At the beginning:

$$\{ \text{lseg } a \cdot \alpha (i, k) \}$$

$$\{ \exists j. i \mapsto a, j * \text{lseg } \alpha (j, k) \}$$

$$\{ \exists j. i + 1 \mapsto j * (i \mapsto a * \text{lseg } \alpha (j, k)) \}$$

$$j := [i + 1];$$

$$\{ i + 1 \mapsto j * (i \mapsto a * \text{lseg } \alpha (j, k)) \}$$

**dispose** i ;

$$\{ i + 1 \mapsto j * \text{lseg } \alpha (j, k) \}$$

**dispose** i + 1 ;

$$\{ \text{lseg } \alpha (j, k) \}$$

$$i := j$$

$$\{ \text{lseg } \alpha (i, k) \}$$

- At the end of a nonempty segment:

$$\{ \text{lseg } \alpha (i, j) * j \mapsto a, k * k \mapsto b, l \}$$

$$[j + 1] := l;$$

$$\{ \text{lseg } \alpha (i, j) * j \mapsto a, l * k \mapsto b, l \}$$

**dispose** k ; **dispose** k + 1

$$\{ \text{lseg } \alpha (i, j) * j \mapsto a, l \}$$

$$\{ \text{lseg } \alpha \cdot a (i, l) \}$$

## A Cyclic Buffer

$$\exists \beta. (\text{lseg } \alpha (i, j) * \text{lseg } \beta (j, i)) \wedge m = \# \alpha \wedge n = \# \alpha + \# \beta$$

When  $i = j$ , the buffer is either empty ( $\# \alpha = 0$ ) or full ( $\# \beta = 0$ ).

To insert an element:

$$\{\exists \beta. (\text{lseg } \alpha (i, j) * \text{lseg } \beta (j, i)) \wedge m = \# \alpha \wedge n = \# \alpha + \# \beta \wedge n - m > 0\}$$

$$\{\exists b, \beta. (\text{lseg } \alpha (i, j) * \text{lseg } b \cdot \beta (j, i)) \wedge m = \# \alpha \wedge n = \# \alpha + \# b \cdot \beta\}$$

$$\{\exists \beta, j''. (\text{lseg } \alpha (i, j) * j \mapsto -, j'' * \text{lseg } \beta (j'', i)) \wedge m = \# \alpha \wedge n - 1 = \# \alpha + \# \beta\}$$

$[j] := x;$

$$\{\exists \beta, j''. (\text{lseg } \alpha (i, j) * j \mapsto x, j'' * \text{lseg } \beta (j'', i)) \wedge m = \# \alpha \wedge n - 1 = \# \alpha + \# \beta\}$$

$$\{\exists \beta, j''. j + 1 \mapsto j'' * ((\text{lseg } \alpha (i, j) * j \mapsto x * \text{lseg } \beta (j'', i)) \wedge m = \# \alpha \wedge n - 1 = \# \alpha + \# \beta)\}$$

$j := [j + 1];$

$$\{\exists \beta, j'. j' + 1 \mapsto j * ((\text{lseg } \alpha (i, j') * j' \mapsto x * \text{lseg } \beta (j, i)) \wedge m = \# \alpha \wedge n - 1 = \# \alpha + \# \beta)\}$$

$$\{\exists \beta, j'. (\text{lseg } \alpha (i, j') * j' \mapsto x, j * \text{lseg } \beta (j, i)) \wedge m = \# \alpha \wedge n - 1 = \# \alpha + \# \beta\}$$

$$\{\exists \beta. (\text{lseg } \alpha \cdot x (i, j) * \text{lseg } \beta (j, i)) \wedge m + 1 = \# \alpha \cdot x \wedge n = \# \alpha \cdot x + \# \beta\}$$

$m := m + 1$

$$\{\exists \beta. (\text{lseg } \alpha \cdot x (i, j) * \text{lseg } \beta (j, i)) \wedge m = \# \alpha \cdot x \wedge n = \# \alpha \cdot x + \# \beta\}$$

## A Cyclic Buffer (continued)

Note the above use of (LKG) for  $j := [j+1]$ , with  $r$  as  $((\text{lseg } \alpha (i, j') * j' \mapsto x * \text{lseg } \beta (j'', i)) \wedge m = \# \alpha \wedge n - 1 = \# \alpha + \# \beta)$ .

To remove an element:

$$\{\exists \beta. (\text{lseg } a \cdot \alpha (i, j) * \text{lseg } \beta (j, i)) \wedge$$

$$m = \# a \cdot \alpha \wedge n = \# a \cdot \alpha + \# \beta \wedge m > 0\}$$

$$\{\exists \beta, i''. (i \mapsto a, i'' * \text{lseg } \alpha (i'', j) * \text{lseg } \beta (j, i)) \wedge$$

$$m - 1 = \# \alpha \wedge n - 1 = \# \alpha + \# \beta\}$$

$$\{\exists \beta, i'', y. (i \mapsto y, i'' * \text{lseg } \alpha (i'', j) * \text{lseg } \beta (j, i)) \wedge$$

$$m - 1 = \# \alpha \wedge n - 1 = \# \alpha + \# \beta \wedge y = a\}$$

$$y := [i];$$

$$\{\exists \beta, i''. (i \mapsto y, i'' * \text{lseg } \alpha (i'', j) * \text{lseg } \beta (j, i)) \wedge$$

$$m - 1 = \# \alpha \wedge n - 1 = \# \alpha + \# \beta \wedge y = a\}$$

$$i := [i + 1];$$

$$\{\exists \beta, i'. (i' \mapsto y, i * \text{lseg } \alpha (i, j) * \text{lseg } \beta (j, i')) \wedge$$

$$m - 1 = \# \alpha \wedge n - 1 = \# \alpha + \# \beta \wedge y = a\}$$

$$\{\exists \beta. (\text{lseg } \alpha (i, j) * \text{lseg } \beta \cdot y (j, i)) \wedge$$

$$m - 1 = \# \alpha \wedge n = \# \alpha + \# \beta \cdot y \wedge y = a\}$$

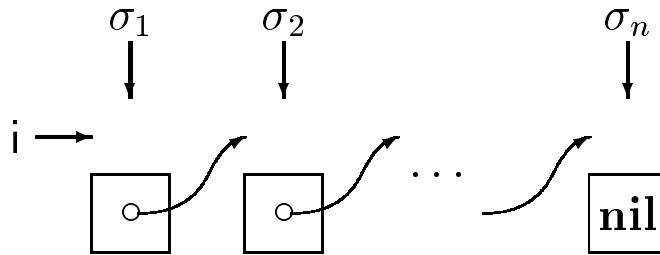
$$\{\exists \beta. (\text{lseg } \alpha (i, j) * \text{lseg } \beta (j, i)) \wedge$$

$$m - 1 = \# \alpha \wedge n = \# \alpha + \# \beta \wedge y = a\}$$

$$m := m - 1$$

$$\{\exists \beta. (\text{lseg } \alpha (i, j) * \text{lseg } \beta (j, i)) \wedge m = \# \alpha \wedge n = \# \alpha + \# \beta \wedge y = a\}$$

## Bornat Lists

listN  $\sigma$  i:

is defined by

$$\text{listN } \epsilon i \stackrel{\text{def}}{=} \mathbf{emp} \wedge i = \mathbf{nil}$$

$$\text{listN } (a \cdot \sigma) i \stackrel{\text{def}}{=} a = i \wedge \exists j. i + 1 \mapsto j * \text{listN } \sigma j.$$

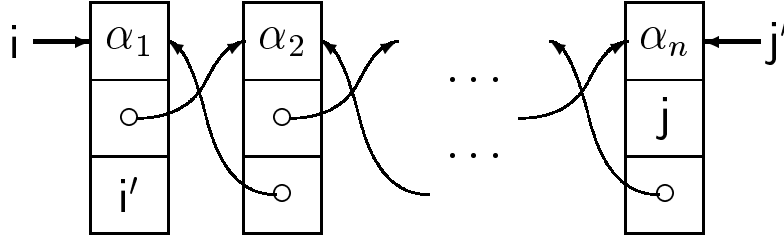
Similarly, one can define Bornat list segments and nontouching Bornat list segments.

## Reversing a Bornat List

$$\begin{aligned}
& \{\text{listN } \sigma_0 \text{ } i\} \\
& \{\text{listN } \sigma_0 \text{ } i * (\mathbf{emp} \wedge \mathbf{nil} = \mathbf{nil})\} \\
& \mathbf{j} := \mathbf{nil}; \\
& \{\text{listN } \sigma_0 \text{ } i * (\mathbf{emp} \wedge \mathbf{j} = \mathbf{nil})\} \\
& \{\text{listN } \sigma_0 \text{ } i * \text{listN } \epsilon \text{ } \mathbf{j}\} \\
& \{\exists \sigma, \tau. (\text{listN } \sigma \text{ } i * \text{listN } \tau \text{ } \mathbf{j}) \wedge \sigma_0^\dagger = \sigma^\dagger \cdot \tau\} \\
& \mathbf{while } i \neq \mathbf{nil} \text{ do} \\
& \quad \left( \{\exists \sigma, \tau. (\text{listN } (i \cdot \sigma) \text{ } i * \text{listN } \tau \text{ } \mathbf{j}) \wedge \sigma_0^\dagger = (i \cdot \sigma)^\dagger \cdot \tau\} \right. \\
& \quad \quad \{\exists \sigma, \tau, k. (i + 1 \mapsto k * \text{listN } \sigma \text{ } k * \text{listN } \tau \text{ } \mathbf{j}) \\
& \quad \quad \quad \wedge \sigma_0^\dagger = (i \cdot \sigma)^\dagger \cdot \tau\} \\
& \quad \quad \mathbf{k} := [i + 1]; \\
& \quad \quad \{\exists \sigma, \tau. (i + 1 \mapsto k * \text{listN } \sigma \text{ } k * \text{listN } \tau \text{ } \mathbf{j}) \\
& \quad \quad \quad \wedge \sigma_0^\dagger = (i \cdot \sigma)^\dagger \cdot \tau\} \\
& \quad \quad [i + 1] := \mathbf{j}; \\
& \quad \quad \{\exists \sigma, \tau. (i + 1 \mapsto \mathbf{j} * \text{listN } \sigma \text{ } k * \text{listN } \tau \text{ } \mathbf{j}) \\
& \quad \quad \quad \wedge \sigma_0^\dagger = (i \cdot \sigma)^\dagger \cdot \tau\} \\
& \quad \quad \{\exists \sigma, \tau. (\text{listN } \sigma \text{ } k * \text{listN } (i \cdot \tau) \text{ } i) \wedge \sigma_0^\dagger = \sigma^\dagger \cdot i \cdot \tau\} \\
& \quad \quad \{\exists \sigma, \tau. (\text{listN } \sigma \text{ } k * \text{listN } \tau \text{ } i) \wedge \sigma_0^\dagger = \sigma^\dagger \cdot \tau\} \\
& \quad \quad \mathbf{j} := i; i := k \\
& \quad \quad \left. \{\exists \sigma, \tau. (\text{listN } \sigma \text{ } i * \text{listN } \tau \text{ } \mathbf{j}) \wedge \sigma_0^\dagger = \sigma^\dagger \cdot \tau\} \right) \\
& \{\exists \sigma, \tau. \text{listN } \tau \text{ } \mathbf{j} \wedge \sigma_0^\dagger = \sigma^\dagger \cdot \tau \wedge \sigma = \epsilon\} \\
& \{\text{listN } \sigma_0^\dagger \text{ } \mathbf{j}\}
\end{aligned}$$

## Doubly-Linked List Segments

$\text{dlseg } \alpha (i, i', j, j')$ :



is defined by

$$\text{dlseg } \epsilon (i, i', j, j') \stackrel{\text{def}}{=} \mathbf{emp} \wedge i = j \wedge i' = j'$$

$$\text{dlseg } a \cdot \alpha (i, i', k, k') \stackrel{\text{def}}{=} \exists j. i \mapsto a, j, i' * \text{dlseg } \alpha (j, i, k, k'),$$

## Properties

$$\text{dlseg } a (i, i', j, j') \Leftrightarrow i \mapsto a, j, i' \wedge i = j'$$

$$\text{dlseg } \alpha \cdot \beta (i, i', k, k') \Leftrightarrow \exists j, j'. \text{dlseg } \alpha (i, i', j, j') * \text{dlseg } \beta (j, j', k, k')$$

$$\text{dlseg } \alpha \cdot b (i, i', k, k') \Leftrightarrow \exists j'. \text{dlseg } \alpha (i, i', k', j') * k' \mapsto b, k, j'.$$

## Emptiness Conditions

$$\text{dlseg } \alpha (i, i', j, j') \wedge i = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge j = \mathbf{nil} \wedge i' = j')$$

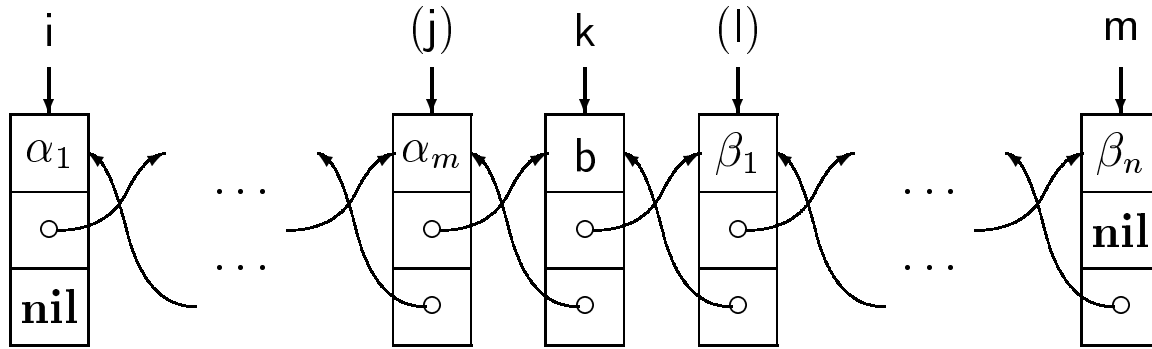
$$\text{dlseg } \alpha (i, i', j, j') \wedge j' = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge i' = \mathbf{nil} \wedge i = j)$$

$$\text{dlseg } \alpha (i, i', j, j') \wedge i \neq j \Rightarrow \alpha \neq \epsilon$$

$$\text{dlseg } \alpha (i, i', j, j') \wedge i' \neq j' \Rightarrow \alpha \neq \epsilon.$$

(One can also define doubly-linked lists and nontouching segments.)

## Deleting an Element from a Doubly-Linked List



$$\{\exists j, l. \text{dlseg } \alpha (i, \mathbf{nil}, k, j) * k \mapsto b, l, j * \text{dlseg } \beta (l, k, \mathbf{nil}, m)\}$$

$$l := [k + 1] ; j := [k + 2] ;$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, k, j) * k \mapsto b, l, j * \text{dlseg } \beta (l, k, \mathbf{nil}, m)\}$$

**dispose k ; dispose k + 1 ; dispose k + 2 ;**

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, k, j) * \text{dlseg } \beta (l, k, \mathbf{nil}, m)\}$$

**if j = nil then**

$$\{\text{dlseg } \beta (l, k, \mathbf{nil}, m) \wedge i = k \wedge \mathbf{nil} = j \wedge \alpha = \epsilon\}$$

$$i := l$$

$$\{\text{dlseg } \beta (l, k, \mathbf{nil}, m) \wedge i = l \wedge \mathbf{nil} = j \wedge \alpha = \epsilon\}$$

**else**

$$\{\exists \alpha', a, n. (\text{dlseg } \alpha' (i, \mathbf{nil}, j, n) * j \mapsto a, k, n$$

$$* \text{dlseg } \beta (l, k, \mathbf{nil}, m)) \wedge \alpha = \alpha' \cdot a\}$$

$$[j + 1] := l ;$$

$$\{\exists \alpha', a, n. (\text{dlseg } \alpha' (i, \mathbf{nil}, j, n) * j \mapsto a, l, n$$

$$* \text{dlseg } \beta (l, k, \mathbf{nil}, m)) \wedge \alpha = \alpha' \cdot a\}$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, l, j) * \text{dlseg } \beta (l, k, \mathbf{nil}, m)\}$$

$$\vdots$$

## Deleting from a Doubly-Linked List (continued)

$$\vdots$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, l, j) * \text{dlseg } \beta (l, k, \mathbf{nil}, m)\}$$

**if**  $l = \mathbf{nil}$  **then**

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, l, j) \wedge l = \mathbf{nil} \wedge k = m \wedge \beta = \epsilon\}$$

$$m := j$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, l, j) \wedge l = \mathbf{nil} \wedge j = m \wedge \beta = \epsilon\}$$

**else**

$$\{\exists a, \beta', n. (\text{dlseg } \alpha (i, \mathbf{nil}, l, j) * l \mapsto a, n, k$$

$$* \text{dlseg } \beta' (n, l, \mathbf{nil}, m)) \wedge \beta = a \cdot \beta'\}$$

$$[l + 2] := j$$

$$\{\exists a, \beta', n. (\text{dlseg } \alpha (i, \mathbf{nil}, l, j) * l \mapsto a, n, j$$

$$* \text{dlseg } \beta' (n, l, \mathbf{nil}, m)) \wedge \beta = a \cdot \beta'\}$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, l, j) * \text{dlseg } \beta (l, j, \mathbf{nil}, m)\}$$

$$\{\text{dlseg } \alpha \cdot \beta (i, \mathbf{nil}, \mathbf{nil}, m)\}$$

## Inference for Simple Procedures

A simple procedure definition has the form

$$h(x_1, \dots, x_m; y_1, \dots, y_n) = c,$$

where  $y_1, \dots, y_n$  are the free variables modified by  $c$ , and  $x_1, \dots, x_m$  are the other free variables of  $c$ .

If one can prove the specification  $\{p\} c \{q\}$ , then one can assume the specification  $\{p\} h(x_1, \dots, x_m; y_1, \dots, y_n) \{q\}$  in proving a specification of a command that contains calls of  $h$ :

### Simple Procedures (SPROC)

When  $h(x_1, \dots, x_m; y_1, \dots, y_n) = c$ ,

$$\frac{\{p\} c \{q\}}{\{p\} h(x_1, \dots, x_m; y_1, \dots, y_n) \{q\}}.$$

From the conclusion of this rule, one can reason about other calls by using the rule for free variable substitution (FVS), and

$$\text{MD}(h(x_1, \dots, x_m; y_1, \dots, y_n)) = y_1, \dots, y_n.$$

For partial correctness, if  $c$  contains recursive calls, one can assume  $\{p\} h(x_1, \dots, x_m; y_1, \dots, y_n) \{q\}$  in proving  $\{p\} c \{q\}$ :

### Simple Recursive Procedures (SRPROC)

When  $h(x_1, \dots, x_m; y_1, \dots, y_n) = c$ ,

$$\frac{\begin{array}{c} \{p\} h(x_1, \dots, x_m; y_1, \dots, y_n) \{q\} \\ \vdots \\ \{p\} c \{q\} \end{array}}{\{p\} h(x_1, \dots, x_m; y_1, \dots, y_n) \{q\}}.$$

## A Lookup-Pointer Procedure for Doubly-Linked Lists

Since

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, j_0, j')\}$$

**if**  $j' = \mathbf{nil}$  **then**

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, j_0, j') \wedge i = j_0\}$$

$$j := i$$

**else**

$$\{\exists \alpha', \mathbf{b}, \mathbf{k}'. \alpha = \alpha' \cdot \mathbf{b} \wedge (\text{dlseg } \alpha' (i, \mathbf{nil}, j', \mathbf{k}') * j' \mapsto \mathbf{b}, j_0, \mathbf{k}')\}$$

$$j := [j' + 1]$$

$$\{\exists \alpha', \mathbf{b}, \mathbf{k}'. \alpha = \alpha' \cdot \mathbf{b} \wedge$$

$$(\text{dlseg } \alpha' (i, \mathbf{nil}, j', \mathbf{k}') * j' \mapsto \mathbf{b}, j_0, \mathbf{k}') \wedge j = j_0\}$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, j_0, j') \wedge j = j_0\},$$

the procedure

$$\text{lookuprpt}(i, j'; j) =$$

$$\mathbf{if } j' = \mathbf{nil} \mathbf{ then } j := i \mathbf{ else } j := [j' + 1]$$

satisfies

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, j_0, j')\}$$

$$\text{lookuprpt}(i, j'; j)$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, j_0, j') \wedge j = j_0\}.$$

## A Set-Pointer Procedure for Doubly-Linked Lists

Since

$$\{\exists j. \text{dlseg } \alpha (i, \mathbf{nil}, j, j')\}$$

**if**  $j' = \mathbf{nil}$  **then**

$$\{\alpha = \epsilon \wedge \mathbf{emp} \wedge j' = \mathbf{nil}\}$$

$$i := j$$

$$\{\alpha = \epsilon \wedge \mathbf{emp} \wedge j' = \mathbf{nil} \wedge i = j\}$$

**else**

$$\{\exists \alpha', b, j, k'. \alpha = \alpha' \cdot b \wedge (\text{dlseg } \alpha' (i, \mathbf{nil}, j', k') * j' \mapsto b, j, k')\}$$

$$[j' + 1] := j$$

$$\{\exists \alpha', b, k'. \alpha = \alpha' \cdot b \wedge (\text{dlseg } \alpha' (i, \mathbf{nil}, j', k') * j' \mapsto b, j, k')\}$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, j, j')\},$$

the procedure

$$\text{setrpt}(j, j'; i) =$$

$$\mathbf{if } j' = \mathbf{nil} \mathbf{ then } i := j \mathbf{ else } [j' + 1] := j$$

satisfies

$$\{\exists j. \text{dlseg } \alpha (i, \mathbf{nil}, j, j')\}$$

$$\text{setrpt}(j, j'; i)$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, j, j')\}.$$

## Upon Reflection

The procedure

$$\text{lookuplpt}(i, j'; i') =$$

$$\text{if } i = \mathbf{nil} \text{ then } i' := j' \text{ else } i' := [i + 2]$$

satisfies

$$\{\text{dlseg } \alpha(i, i'_0, \mathbf{nil}, j')\}$$

$$\text{lookuplpt}(i, j'; i')$$

$$\{\text{dlseg } \alpha(i, i'_0, \mathbf{nil}, j') \wedge i' = i'_0\},$$

and the procedure

$$\text{setlpt}(i, i'; j') =$$

$$\text{if } i = \mathbf{nil} \text{ then } j' := i' \text{ else } [i + 2] := i'$$

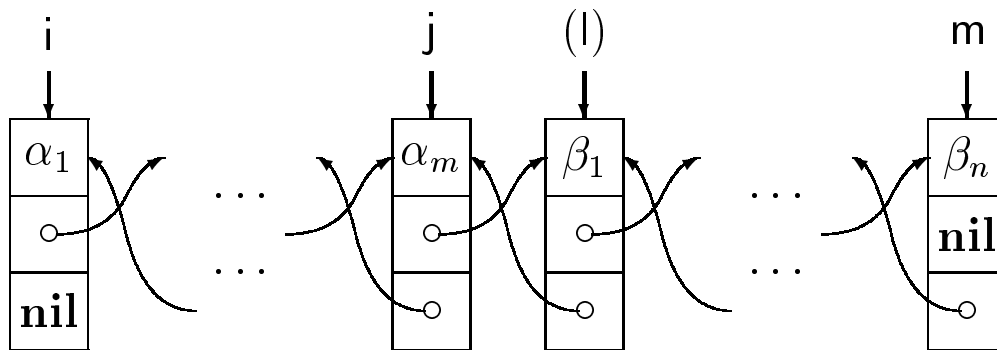
satisfies

$$\{\exists i'. \text{dlseg } \alpha(i, i', \mathbf{nil}, j')\}$$

$$\text{setlpt}(i, i'; j')$$

$$\{\text{dlseg } \alpha(i, i', \mathbf{nil}, j')\}.$$

## Inserting an Element into a Doubly-Linked List



$$\{\exists l. \text{dlseg } \alpha (i, \mathbf{nil}, l, j) * \text{dlseg } \beta (l, j, \mathbf{nil}, m)\}$$

$$\text{lookuprpt}(i, j; l) ;$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, l, j) * \text{dlseg } \beta (l, j, \mathbf{nil}, m)\}$$

$$k := \mathbf{cons}(a, l, j) ;$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, l, j) * k \mapsto a, l, j * \text{dlseg } \beta (l, j, \mathbf{nil}, m)\}$$

$$\text{setrpt}(k, j; i) ;$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, k, j) * k \mapsto a, l, j * \text{dlseg } \beta (l, j, \mathbf{nil}, m)\}$$

$$\text{setlpt}(l, k; m)$$

$$\{\text{dlseg } \alpha (i, \mathbf{nil}, k, j) * k \mapsto a, l, j * \text{dlseg } \beta (l, k, \mathbf{nil}, m)\}$$

$$\{\text{dlseg } \alpha \cdot a \cdot \beta (i, \mathbf{nil}, \mathbf{nil}, m)\}$$

## Deriving the Specification of the Call of `lookuprpt`

From:

$$\begin{aligned} & \{\text{dlseg } \alpha (i, \mathbf{nil}, j_0, j')\} \\ & \text{lookuprpt}(i, j'; j) \\ & \{\text{dlseg } \alpha (i, \mathbf{nil}, j_0, j') \wedge j = j_0\}, \end{aligned}$$

using Free Variable Substitution (FVS):

$$\begin{aligned} & \{\text{dlseg } \alpha (i, \mathbf{nil}, j_0, j)\} \\ & \text{lookuprpt}(i, j; l) \\ & \{\text{dlseg } \alpha (i, \mathbf{nil}, j_0, j) \wedge l = j_0\}, \end{aligned}$$

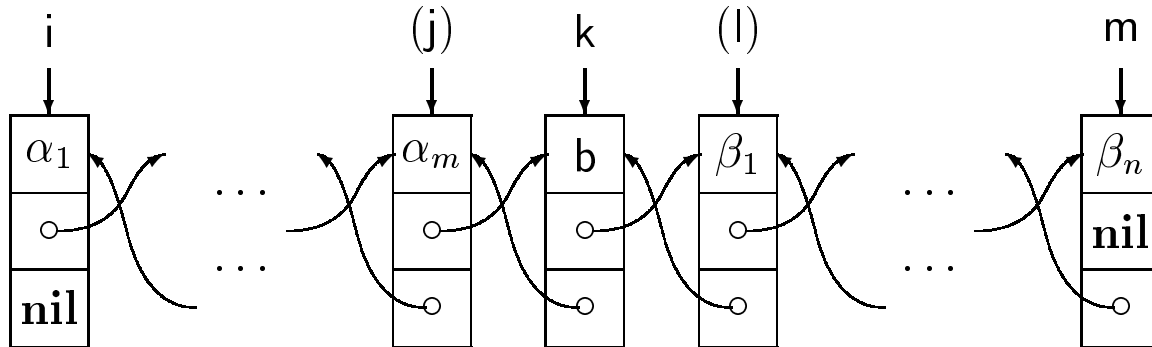
using the Frame Rule (FR) and an obvious property of equality:

$$\begin{aligned} & \{\text{dlseg } \alpha (i, \mathbf{nil}, j_0, j) * \text{dlseg } \beta (j_0, j, \mathbf{nil}, m)\} \\ & \text{lookuprpt}(i, j; l) \\ & \{(\text{dlseg } \alpha (i, \mathbf{nil}, j_0, j) * \text{dlseg } \beta (j_0, j, \mathbf{nil}, m)) \wedge l = j_0\} \\ & \{\text{dlseg } \alpha (i, \mathbf{nil}, l, j) * \text{dlseg } \beta (l, j, \mathbf{nil}, m)\}, \end{aligned}$$

using the rules for Existential Quantification (EQ), Renaming (RN), and the elimination of a vacuous quantifier:

$$\begin{aligned} & \{\exists l. \text{dlseg } \alpha (i, \mathbf{nil}, l, j) * \text{dlseg } \beta (l, j, \mathbf{nil}, m)\} \\ & \{\exists j_0. \text{dlseg } \alpha (i, \mathbf{nil}, j_0, j) * \text{dlseg } \beta (j_0, j, \mathbf{nil}, m)\} \\ & \text{lookuprpt}(i, j; l) \\ & \{\exists j_0. \text{dlseg } \alpha (i, \mathbf{nil}, l, j) * \text{dlseg } \beta (l, j, \mathbf{nil}, m)\} \\ & \{\text{dlseg } \alpha (i, \mathbf{nil}, l, j) * \text{dlseg } \beta (l, j, \mathbf{nil}, m)\}. \end{aligned}$$

## Deleting an Element from a Doubly-Linked List



$$\{\exists j, l. \text{dlseg } \alpha(i, \mathbf{nil}, k, j) * k \mapsto b, l, j * \text{dlseg } \beta(l, k, \mathbf{nil}, m)\}$$

$$l := [k + 1] ; j := [k + 2] ;$$

$$\{\text{dlseg } \alpha(i, \mathbf{nil}, k, j) * k \mapsto b, l, j * \text{dlseg } \beta(l, k, \mathbf{nil}, m)\}$$

**dispose**  $k$  ; **dispose**  $k + 1$  ; **dispose**  $k + 2$  ;

$$\{\text{dlseg } \alpha(i, \mathbf{nil}, k, j) * \text{dlseg } \beta(l, k, \mathbf{nil}, m)\}$$

**setrpt**( $l, j; i$ ) ;

$$\{\text{dlseg } \alpha(i, \mathbf{nil}, l, j) * \text{dlseg } \beta(l, k, \mathbf{nil}, m)\}$$

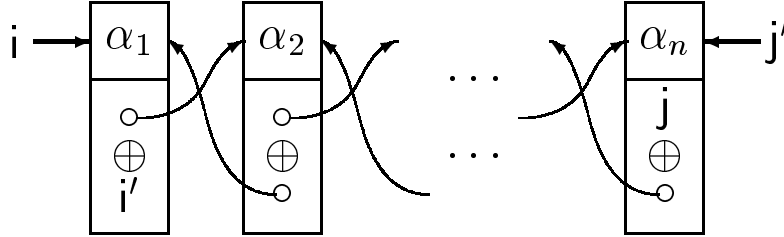
**setlpt**( $l, j; m$ )

$$\{\text{dlseg } \alpha(i, \mathbf{nil}, l, j) * \text{dlseg } \beta(l, j, \mathbf{nil}, m)\}$$

$$\{\text{dlseg } \alpha \cdot \beta(i, \mathbf{nil}, \mathbf{nil}, m)\}$$

## An Inductive Definition: Xor-Linked List Segments

$\text{xlseg } \alpha (i, i', j, j')$ :



is defined by

$$\text{xlseg } \epsilon (i, i', j, j') \stackrel{\text{def}}{=} \mathbf{emp} \wedge i = j \wedge i' = j'$$

$$\text{xlseg } a \cdot \alpha (i, i', k, k') \stackrel{\text{def}}{=} \exists j. i \mapsto a, (j \oplus i') * \text{xlseg } \alpha (j, i, k, k').$$

## Properties

$$\text{xlseg } a (i, i', j, j') \Leftrightarrow i \mapsto a, (j \oplus i') \wedge i = j'$$

$$\text{xlseg } \alpha \cdot \beta (i, i', k, k') \Leftrightarrow \exists j, j'. \text{xlseg } \alpha (i, i', j, j') * \text{xlseg } \beta (j, j', k, k')$$

$$\text{xlseg } \alpha \cdot b (i, i', k, k') \Leftrightarrow \exists j'. \text{xlseg } \alpha (i, i', k', j') * k' \mapsto b, (k \oplus j').$$

## Emptyness Conditions (as with dlseg)

$$\text{xlseg } \alpha (i, i', j, j') \wedge i = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge j = \mathbf{nil} \wedge i' = j')$$

$$\text{xlseg } \alpha (i, i', j, j') \wedge j' = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge i' = \mathbf{nil} \wedge i = j)$$

$$\text{xlseg } \alpha (i, i', j, j') \wedge i \neq j \Rightarrow \alpha \neq \epsilon$$

$$\text{xlseg } \alpha (i, i', j, j') \wedge i' \neq j' \Rightarrow \alpha \neq \epsilon.$$

## A Set-Pointer Procedure for Xor-linked Lists

Since

$$\{\text{xlse}g \alpha (i, \mathbf{nil}, j, j')\}$$

**if**  $j' = \mathbf{nil}$  **then**

$$\{\alpha = \epsilon \wedge \mathbf{emp} \wedge j' = \mathbf{nil}\}$$

$$i := k$$

$$\{\alpha = \epsilon \wedge \mathbf{emp} \wedge j' = \mathbf{nil} \wedge i = k\}$$

**else**

$$\{\exists \alpha', b, k'. \alpha = \alpha' \cdot b \wedge (\text{xlse}g \alpha' (i, \mathbf{nil}, j', k') * j' \mapsto b, (j \oplus k'))\}$$

$$\mathbf{newvar} \ x \ \mathbf{in} \ (x := [j' + 1] ; [j' + 1] := x \oplus j \oplus k)$$

$$\{\exists \alpha', b, k'. \alpha = \alpha' \cdot b \wedge (\text{xlse}g \alpha' (i, \mathbf{nil}, j', k') * j' \mapsto b, (k \oplus k'))\}$$

$$\{\text{xlse}g \alpha (i, \mathbf{nil}, k, j')\},$$

the procedure

$$\text{xsetrpt}(j, j', k; i) =$$

$$\mathbf{if} \ j' = \mathbf{nil} \ \mathbf{then} \ i := k \ \mathbf{else}$$

$$\mathbf{newvar} \ x \ \mathbf{in} \ (x := [j' + 1] ; [j' + 1] := x \oplus j \oplus k)$$

satisfies

$$\{\text{xlse}g \alpha (i, \mathbf{nil}, j, j')\}$$

$$\text{xsetrpt}(j, j', k; i)$$

$$\{\text{xlse}g \alpha (i, \mathbf{nil}, k, j')\}.$$

## Upon Reflection

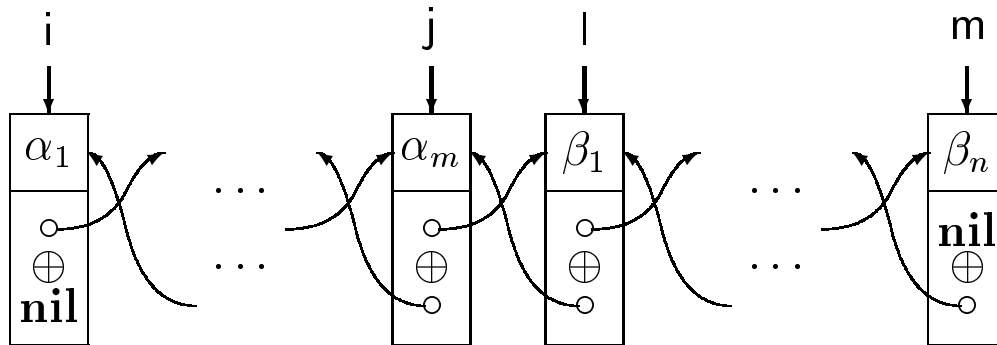
The procedure

```
xsetlpt(i, i', k; j') =
  if i = nil then j' := k else
    newvar x in (x := [i + 1] ; [i + 1] := x ⊕ i' ⊕ k)
```

satisfies

```
{xlseg α (i, i', nil, j')}
xsetlpt(i, i', k; j')
{xlseg α (i, k, nil, j')}.
```

## Inserting an Element into a Xor-Linked List



$$\{\text{xlseg } \alpha (i, \mathbf{nil}, l, j) * \text{xlseg } \beta (l, j, \mathbf{nil}, m)\}$$

$$k := \mathbf{cons}(a, l \oplus j) ;$$

$$\{\text{xlseg } \alpha (i, \mathbf{nil}, l, j) * k \mapsto a, (l \oplus j) * \text{xlseg } \beta (l, j, \mathbf{nil}, m)\}$$

$$\text{xsetrpt}(l, j, k; i) ;$$

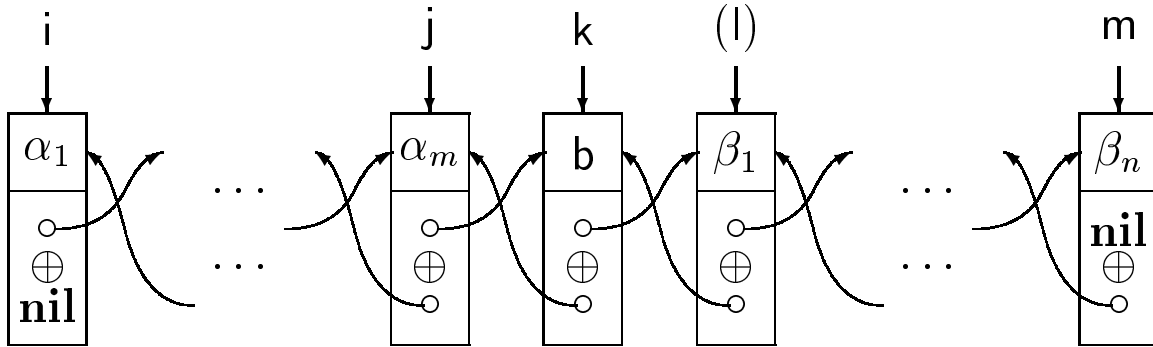
$$\{\text{xlseg } \alpha (i, \mathbf{nil}, k, j) * k \mapsto a, (l \oplus j) * \text{xlseg } \beta (l, j, \mathbf{nil}, m)\}$$

$$\text{xsetlpt}(l, j, k; m)$$

$$\{\text{xlseg } \alpha (i, \mathbf{nil}, k, j) * k \mapsto a, (l \oplus j) * \text{xlseg } \beta (l, k, \mathbf{nil}, m)\}$$

$$\{\text{xlseg } \alpha \cdot a \cdot \beta (i, \mathbf{nil}, \mathbf{nil}, m)\}$$

## Deleting an Element from a Xor-Linked List



$$\{\exists l. \text{xlseg } \alpha (i, \mathbf{nil}, k, j) * k \mapsto b, (l \oplus j) * \text{xlseg } \beta (l, k, \mathbf{nil}, m)\}$$

**newvar**  $x$  **in** ( $x := [k + 1]$  ;  $l := x \oplus j$ ) ;

$$\{\text{xlseg } \alpha (i, \mathbf{nil}, k, j) * k \mapsto b, (l \oplus j) * \text{xlseg } \beta (l, k, \mathbf{nil}, m)\}$$

**dispose**  $k$  ; **dispose**  $k + 1$  ;

$$\{\text{xlseg } \alpha (i, \mathbf{nil}, k, j) * \text{xlseg } \beta (l, k, \mathbf{nil}, m)\}$$

**xsetrpt**( $k, j, l; i$ ) ;

$$\{\text{xlseg } \alpha (i, \mathbf{nil}, l, j) * \text{xlseg } \beta (l, k, \mathbf{nil}, m)\}$$

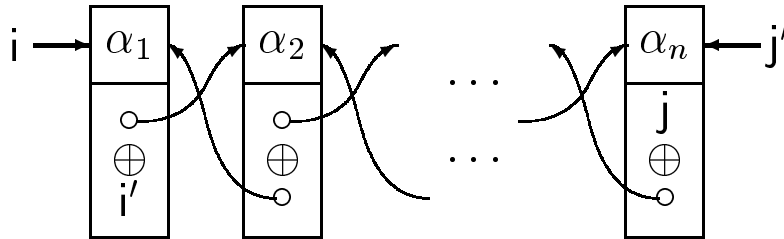
**xsetlpt**( $l, k, j; m$ )

$$\{\text{xlseg } \alpha (i, \mathbf{nil}, l, j) * \text{xlseg } \beta (l, j, \mathbf{nil}, m)\}$$

$$\{\text{xlseg } \alpha \cdot \beta (i, \mathbf{nil}, \mathbf{nil}, m)\}$$

## Reversing an Xor-Linked List Segment

$\text{xlseg } \alpha (i, i', j, j')$ :



$$\text{xlseg } \alpha (i, i', j, j') \Leftrightarrow \text{xlseg } \alpha^\dagger (j', j, i', i)$$

Proof (by induction on  $\alpha$ ):

Base case:

$$\text{xlseg } \epsilon (i, i', j, j') \Leftrightarrow \mathbf{emp} \wedge i = j \wedge i' = j' \Leftrightarrow \text{xlseg } \epsilon^\dagger (j', j, i', i)$$

Induction step:

$$\begin{aligned} & \text{xlseg } a \cdot \alpha (i, i', k, k') \\ & \Leftrightarrow \exists j. i \mapsto a, (j \oplus i') * \text{xlseg } \alpha (j, i, k, k') \\ & \Leftrightarrow \exists j. \text{xlseg } \alpha^\dagger (k', k, i, j) * i \mapsto a, (i' \oplus j) \\ & \Leftrightarrow \text{xlseg } \alpha^\dagger \cdot a (k', k, i', i) \\ & \Leftrightarrow \text{xlseg } (a \cdot \alpha)^\dagger (k', k, i', i). \end{aligned}$$

## Images

The *image*  $\{\alpha\}$  of a sequence  $\alpha$  is the set

$$\{\alpha_i \mid 1 \leq i \leq \#\alpha\}$$

of values occurring as components of  $\alpha$ . It satisfies

$$\{\epsilon\} = \{\} \tag{3}$$

$$\{[x]\} = \{x\} \tag{4}$$

$$\{\alpha \cdot \beta\} = \{\alpha\} \cup \{\beta\} \tag{5}$$

$$\#\{\alpha\} \leq \#\alpha. \tag{6}$$

## Pointwise Extension of Relations

If  $\rho$  is a binary relation between values (e.g., integers), then  $\rho^*$  is the binary relation between sets of values such that

$$S \rho^* T \text{ iff } \forall x \in S. \forall y \in T. x \rho y.$$

Pointwise extension satisfies:

$$S' \subseteq S \wedge S \rho^* T \Rightarrow S' \rho^* T \quad (7)$$

$$T' \subseteq T \wedge S \rho^* T \Rightarrow S \rho^* T' \quad (8)$$

$$\{\} \rho^* T \quad (9)$$

$$S \rho^* \{\} \quad (10)$$

$$\{x\} \rho^* \{y\} \Leftrightarrow x \rho y \quad (11)$$

$$(S \cup S') \rho^* T \Leftrightarrow S \rho^* T \wedge S' \rho^* T \quad (12)$$

$$S \rho^* (T \cup T') \Leftrightarrow S \rho^* T \wedge S \rho^* T'. \quad (13)$$

## Some Abbreviations

$$x \rho^* T \stackrel{\text{def}}{=} \{x\} \rho^* T \quad S \rho^* y \stackrel{\text{def}}{=} S \rho^* \{y\}$$

## Ordering

We write **ord**  $\alpha$  if the sequence  $\alpha$  is ordered in nonstrict increasing order. Then

$$\#\alpha \leq 1 \Rightarrow \mathbf{ord} \alpha \quad (14)$$

$$\mathbf{ord} \alpha \cdot \beta \Leftrightarrow \mathbf{ord} \alpha \wedge \mathbf{ord} \beta \wedge \{\alpha\} \leq^* \{\beta\}. \quad (15)$$

## Rearrangement

We say that a sequence  $\beta$  is a *rearrangement* of a sequence  $\alpha$ , written  $\beta \sim \alpha$ , iff  $\#\beta = \#\alpha$  and there is a permutation  $\phi$  such that

$$\forall k. 1 \leq k \leq \#\alpha \text{ implies } \beta_k = \alpha_{\phi(k)}.$$

Then

$$\alpha \sim \alpha \quad (16)$$

$$\alpha \sim \beta \Rightarrow \beta \sim \alpha \quad (17)$$

$$\alpha \sim \beta \wedge \beta \sim \gamma \Rightarrow \alpha \sim \gamma \quad (18)$$

$$\alpha \sim \alpha' \wedge \beta \sim \beta' \Rightarrow \alpha \cdot \beta \sim \alpha' \cdot \beta' \quad (19)$$

$$\alpha \cdot \beta \sim \beta \cdot \alpha \quad (20)$$

$$\alpha \sim \beta \Rightarrow \{\alpha\} = \{\beta\}. \quad (21)$$

## Specifications for Sorting by Merging

We define the abbreviation

$$\text{lseg } \alpha (e, -) \stackrel{\text{def}}{=} \exists x. \text{lseg } \alpha (e, x).$$

Then we will define procedures satisfying

$$\begin{aligned} & \{ \text{lseg } \alpha (i, j_0) \wedge \# \alpha = n \wedge n \geq 1 \} \\ & \text{mergesort}(n; i, j) \\ & \{ \exists \beta. \text{lseg } \beta (i, -) \wedge \beta \sim \alpha \wedge \mathbf{ord} \beta \wedge j = j_0 \}. \end{aligned}$$

and

$$\begin{aligned} & \{ (\text{lseg } \beta_1 (i_1, -) \wedge \mathbf{ord} \beta_1 \wedge \# \beta_1 = n_1 \wedge n_1 \geq 1) \\ & \quad * (\text{lseg } \beta_2 (i_2, -) \wedge \mathbf{ord} \beta_2 \wedge \# \beta_2 = n_2 \wedge n_2 \geq 1) \} \\ & \text{merge}(n_1, n_2, i_1, i_2; i) \\ & \{ \exists \beta. \text{lseg } \beta (i, -) \wedge \beta \sim \beta_1 \cdot \beta_2 \wedge \mathbf{ord} \beta \}. \end{aligned}$$

## An Arithmetical Note

Suppose  $n \geq 2$ . Then  $2 \leq n \leq 2 \times n - 2$ , and since division by two is monotone:

$$1 = 2 \div 2 \leq n \div 2 \leq (2 \times n - 2) \div 2 = n - 1.$$

Thus if  $n_1 = n \div 2$  and  $n_2 = n - n_1$ , then

$$1 \leq n_1 \leq n - 1 \quad 1 \leq n_2 \leq n - 1 \quad n_1 + n_2 = n.$$

## A Proof for mergesort

$\{\text{lseg } \alpha (i, j_0) \wedge \#\alpha = n \wedge n \geq 1\}$

**if**  $n = 1$  **then**

$\{\text{lseg } \alpha (i, -) \wedge \mathbf{ord} \alpha \wedge i \mapsto -, j_0\}$

$j := [i + 1]$

$\{\text{lseg } \alpha (i, -) \wedge \mathbf{ord} \alpha \wedge j = j_0\}$

**else**

$(n1 := n \div 2 ; n2 := n - n1 ; i1 := i ;$

$\{\exists \alpha_1, \alpha_2, i_2. ((\text{lseg } \alpha_1 (i1, i_2) \wedge \#\alpha_1 = n1 \wedge n1 \geq 1)$

$* (\text{lseg } \alpha_2 (i_2, j_0) \wedge \#\alpha_2 = n2 \wedge n2 \geq 1)) \wedge \alpha = \alpha_1 \cdot \alpha_2\}$

$\text{mergesort}(n1; i1, i2) ;$

$\{\exists \alpha_1, \alpha_2, \beta_1. ((\text{lseg } \beta_1 (i1, -) \wedge \beta_1 \sim \alpha_1 \wedge \mathbf{ord} \beta_1 \wedge \#\alpha_1 = n1 \wedge n1 \geq 1)$

$* (\text{lseg } \alpha_2 (i2, j_0) \wedge \#\alpha_2 = n2 \wedge n2 \geq 1)) \wedge \alpha = \alpha_1 \cdot \alpha_2\}$

$\text{mergesort}(n2; i2, j) ;$

$\{\exists \alpha_1, \alpha_2, \beta_1, \beta_2. ((\text{lseg } \beta_1 (i1, -) \wedge \beta_1 \sim \alpha_1 \wedge \mathbf{ord} \beta_1 \wedge \#\alpha_1 = n1$

$\wedge n1 \geq 1)$

$* (\text{lseg } \beta_2 (i2, -) \wedge \beta_2 \sim \alpha_2 \wedge \mathbf{ord} \beta_2 \wedge \#\alpha_2 = n2 \wedge n2 \geq 1))$

$\wedge \alpha = \alpha_1 \cdot \alpha_2 \wedge j = j_0\}$

$\{\exists \beta_1, \beta_2. ((\text{lseg } \beta_1 (i1, -) \wedge \mathbf{ord} \beta_1 \wedge \#\beta_1 = n1 \wedge n1 \geq 1)$

$* (\text{lseg } \beta_2 (i2, -) \wedge \mathbf{ord} \beta_2 \wedge \#\beta_2 = n2 \wedge n2 \geq 1)) \wedge \alpha \sim \beta_1 \cdot \beta_2 \wedge j = j_0\}$

$\text{merge}(n1, n2, i1, i2; i)$

$\{\exists \beta, \beta_1, \beta_2. \text{lseg } \beta (i, -) \wedge \beta \sim \beta_1 \cdot \beta_2 \wedge \mathbf{ord} \beta \wedge \alpha \sim \beta_1 \cdot \beta_2 \wedge j = j_0\}$

$\{\exists \beta. \text{lseg } \beta (i, -) \wedge \beta \sim \alpha \wedge \mathbf{ord} \beta \wedge j = j_0\}$ .

## A Proof for mergesort (continued)

Thus we may define

```
mergesort(n; i, j) =  
  if n = 1 then j := [i + 1] else  
    (n1 := n ÷ 2 ; n2 := n - n1 ; i1 := i ;  
     mergesort(n1; i1, i2) ; mergesort(n2; i2, j) ;  
     merge(n1, n2, i1, i2; i)).
```

## A Proof for merge

Suppose

$$\{\exists \beta, a1, j1, \gamma_1, j2, \gamma_2.$$

$$\begin{aligned} & (\text{lseg } \beta (i, i1) * i1 \mapsto a1, j1 * \text{lseg } \gamma_1 (j1, -) * i2 \mapsto a2, j2 \\ & * \text{lseg } \gamma_2 (j2, -)) \end{aligned}$$

$$\wedge \# \gamma_1 = n1 \wedge \# \gamma_2 = n2 \wedge a1 \leq a2 \wedge \beta \cdot a1 \cdot \gamma_1 \cdot a2 \cdot \gamma_2 \sim \beta_1 \cdot \beta_2$$

$$\wedge \mathbf{ord} (a1 \cdot \gamma_1) \wedge \mathbf{ord} (a2 \cdot \gamma_2) \wedge \mathbf{ord} \beta \wedge \{\beta\} \leq^* \{a1 \cdot \gamma_1\} \cup \{a2 \cdot \gamma_2\}$$

$\text{merge1}(n1, n2, i1, i2, a2)$

$$\{\exists \beta. \text{lseg } \beta (i, -) \wedge \beta \sim \beta_1 \cdot \beta_2 \wedge \mathbf{ord} \beta\}$$

Then

$$\begin{aligned} & \{(\text{lseg } \beta_1 (i1, -) \wedge \mathbf{ord} \beta_1 \wedge \# \beta_1 = n1 \wedge n1 \geq 1) \\ & * (\text{lseg } \beta_2 (i2, -) \wedge \mathbf{ord} \beta_2 \wedge \# \beta_2 = n2 \wedge n2 \geq 1)\} \end{aligned}$$

**newvar a1 in newvar a2 in**  $(a1 := [i1] ; a2 := [i2] ;$   
**if**  $a1 \leq a2$  **then**  $(i := i1 ; \text{merge1}(n1, n2, i1, i2, a2))$   
**else**  $(i := i2 ; \text{merge1}(n2, n1, i2, i1, a1))$  $)$

$$\{\exists \beta. \text{lseg } \beta (i, -) \wedge \beta \sim \beta_1 \cdot \beta_2 \wedge \mathbf{ord} \beta\}.$$

Thus we may define

$\text{merge}(n1, n2, i1, i2; i) =$

**newvar a1 in newvar a2 in**  $(a1 := [i1] ; a2 := [i2] ;$   
**if**  $a1 \leq a2$  **then**  $(i := i1 ; \text{merge1}(n1, n2, i1, i2, a2))$   
**else**  $(i := i2 ; \text{merge1}(n2, n1, i2, i1, a1))$  $)$

## A Proof for merge1

$\{\exists\beta, a1, j1, \gamma_1, j2, \gamma_2.$

$(\text{lseg } \beta (i, i1) * i1 \mapsto a1, j1 * \text{lseg } \gamma_1 (j1, -) * i2 \mapsto a2, j2$   
 $* \text{lseg } \gamma_2 (j2, -))$

$\wedge \#\gamma_1 = n1 \wedge \#\gamma_2 = n2 \wedge a1 \leq a2 \wedge \beta \cdot a1 \cdot \gamma_1 \cdot a2 \cdot \gamma_2 \sim \beta_1 \cdot \beta_2$

$\wedge \mathbf{ord} (a1 \cdot \gamma_1) \wedge \mathbf{ord} (a2 \cdot \gamma_2) \wedge \mathbf{ord} \beta \wedge \{\beta\} \leq^* \{a1 \cdot \gamma_1\} \cup \{a2 \cdot \gamma_2\}$

**if**  $n1 = 0$  **then**  $[i1 + 1] := i2$

$\{\exists\beta, a1, j2, \gamma_2.$

$(\text{lseg } \beta (i, i1) * i1 \mapsto a1, i2 * i2 \mapsto a2, j2 * \text{lseg } \gamma_2 (j2, -))$

$\wedge \#\gamma_2 = n2 \wedge a1 \leq a2 \wedge \beta \cdot a1 \cdot a2 \cdot \gamma_2 \sim \beta_1 \cdot \beta_2$

$\wedge \mathbf{ord} (a2 \cdot \gamma_2) \wedge \mathbf{ord} \beta \wedge \{\beta\} \leq^* \{a1\} \cup \{a2 \cdot \gamma_2\}$

**else newvar**  $j1$  **in newvar**  $a3$  **in**

$(j1 := [i1 + 1] ; a3 := [j1] ;$

$\{\exists\beta, a1, j3, \gamma_3, j2, \gamma_2.$

$(\text{lseg } \beta (i, i1) * i1 \mapsto a1, j1 * j1 \mapsto a3, j3 * \text{lseg } \gamma_3 (j3, -)$   
 $* i2 \mapsto a2, j2 * \text{lseg } \gamma_2 (j2, -))$

$\wedge \#\gamma_3 = n1 - 1 \wedge \#\gamma_2 = n2 \wedge a1 \leq a2 \wedge \beta \cdot a1 \cdot a3 \cdot \gamma_3 \cdot a2 \cdot \gamma_2 \sim \beta_1 \cdot \beta_2$

$\wedge \mathbf{ord} (a1 \cdot a3 \cdot \gamma_3) \wedge \mathbf{ord} (a2 \cdot \gamma_2) \wedge \mathbf{ord} \beta$

$\wedge \{\beta\} \leq^* \{a1 \cdot a3 \cdot \gamma_3\} \cup \{a2 \cdot \gamma_2\}$

**if**  $a3 \leq a2$  **then**  $\text{merge1}(n1 - 1, n2, j1, i2, a2)$

**else**  $([i1 + 1] := i2 ; \text{merge1}(n2, n1 - 1, i2, j1, a3))$

$\{\exists\beta. \text{lseg } \beta (i, -) \wedge \beta \sim \beta_1 \cdot \beta_2 \wedge \mathbf{ord} \beta\}$

A Proof for `merge1` (continued)

Thus we may define

```
merge1(n1, n2, i1, i2, a2) =  
  if n1 = 0 then [i1 + 1] := i2  
  else newvar j1 in newvar a3 in  
    (j1 := [i1 + 1] ; a3 := [j1] ;  
     if a3 ≤ a2 then merge1(n1 - 1, n2, j1, i2, a2)  
     else ([i1 + 1] := i2 ; merge1(n2, n1 - 1, i2, j1, a3)))
```