AN INTRODUCTION TO SEPARATION LOGIC

4. Lists and List Segments

John C. Reynolds
Carnegie Mellon University

February 17, 2011

©2011 John C. Reynolds

—

Notation for Sequences

When α and β are sequences, we write

- ϵ for the empty sequence.
- $[a]$ for the single-element sequence containing a. (We will omit the brackets when a is not a sequence.)
- $\alpha{\cdot}\beta$ for the composition of α followed by β.
- $\alpha^\dagger$ for the reflection of α.
- $\#\alpha$ for the length of α.
- α_i for the ith component of α.

Some Laws for Sequences

$$\alpha{\cdot}\epsilon = \alpha \qquad \epsilon{\cdot}\alpha = \alpha \qquad (\alpha{\cdot}\beta){\cdot}\gamma = \alpha{\cdot}(\beta{\cdot}\gamma)$$

$$\epsilon^\dagger = \epsilon \qquad [a]^\dagger = [a] \qquad (\alpha{\cdot}\beta)^\dagger = \beta^\dagger{\cdot}\alpha^\dagger$$

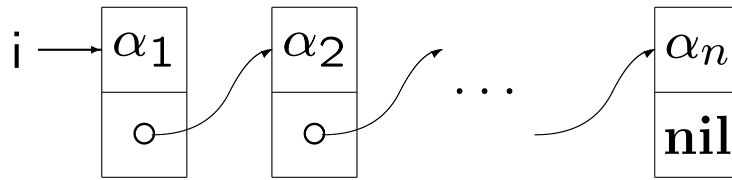
$$\#\epsilon = 0 \qquad \#[a] = 1 \qquad \#(\alpha{\cdot}\beta) = (\#\alpha) + (\#\beta)$$

$$\alpha = \epsilon \vee \exists a, \alpha'.\ \alpha = [a]{\cdot}\alpha' \qquad \alpha = \epsilon \vee \exists \alpha', a.\ \alpha = \alpha'{\cdot}[a].$$

—

Singly-linked Lists

list α i:



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

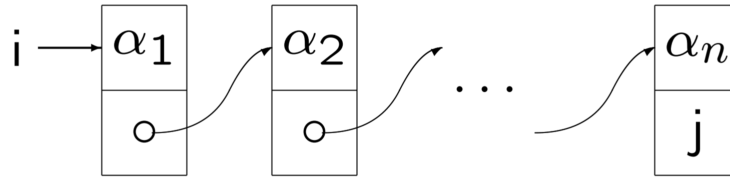
$$\mathsf{list}\ \epsilon\ \mathsf{i} \stackrel{\text{def}}{=} \mathbf{emp} \wedge \mathsf{i} = \mathbf{nil}$$

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

—

Singly-linked List Segments

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



is defined by

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

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

Properties

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

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

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

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

—

Proof of the Composition Property

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

The proof is by induction on the length of α.

When α is empty:

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

When $\alpha = \mathsf{a}{\cdot}\alpha'$:

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

—

Emptyness Conditions

For lists, one can derive a law that shows clearly when a list represents the empty sequence:

$$\mathsf{list}\ \alpha\ \mathsf{i} \Rightarrow (\mathsf{i} = \mathbf{nil} \Leftrightarrow \alpha = \epsilon).$$

For list segments, however, the situation is more complex. One can derive

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

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

—

Nontouching List Segments

When

$$\mathsf{lseg}\ a_1 \cdots a_n\,(\mathsf{i}_0, \mathsf{i}_n),$$

we have

$$\exists \mathsf{i}_1, \ldots \mathsf{i}_{n-1}.\ (\mathsf{i}_0 \mapsto a_1, \mathsf{i}_1) \ * \ (\mathsf{i}_1 \mapsto a_2, \mathsf{i}_2) \ * \ \cdots \ * \ (\mathsf{i}_{n-1} \mapsto a_n, \mathsf{i}_n).$$

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

list segments
- nontouching
- cyclic
  - touching
  - overlapping (forbidden by $*$).

—

Nontouching List Segments (continued)

We can define nontouching list segments inductively by:

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

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

or equivalently, we can define them in terms of lseg:

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

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

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

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

$$\begin{aligned} \mathsf{list}\ \alpha\ \mathsf{i} &\Rightarrow \mathsf{ntlseg}\ \alpha\ (\mathsf{i},\mathbf{nil}) \\ \mathsf{lseg}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{list}\ \beta\ \mathsf{j} &\Rightarrow \mathsf{ntlseg}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{list}\ \beta\ \mathsf{j} \\ \mathsf{lseg}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \hookrightarrow - &\Rightarrow \mathsf{ntlseg}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \hookrightarrow -. \end{aligned}$$

—

Preciseness of List Assertions

The assertions

$$\mathsf{list}\ \alpha\ \mathsf{i} \qquad \mathsf{lseg}\ \alpha\ (\mathsf{i}, \mathsf{j}) \qquad \mathsf{ntlseq}\ \alpha\ (\mathsf{i}, \mathsf{j})$$

are all precise.. On the other hand, although

$$\exists \alpha.\ \mathsf{list}\ \alpha\ \mathsf{i} \qquad \exists \alpha.\ \mathsf{ntlseq}\ \alpha\ (\mathsf{i}, \mathsf{j})$$

are precise,

$$\exists \alpha.\ \mathsf{lseg}\ \alpha\ (\mathsf{i}, \mathsf{j})$$

is not precise.

—

Insertion at the Beginning of a List Segment

$\{\mathsf{lseg}\ \alpha\ (\mathsf{i}, \mathsf{j})\}$

$\mathsf{k} := \mathbf{cons}(\mathsf{a}, \mathsf{i})\ ;$ (CONSNOG)

$\{\mathsf{k} \mapsto \mathsf{a}, \mathsf{i}\ *\ \mathsf{lseg}\ \alpha\ (\mathsf{i}, \mathsf{j})\}$

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

$\{\mathsf{lseg}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{k}, \mathsf{j})\}$

$\mathsf{i} := \mathsf{k}$ (AS)

$\{\mathsf{lseg}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{i}, \mathsf{j})\}$,

or, more concisely:

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

$\mathsf{i} := \mathbf{cons}(\mathsf{a}, \mathsf{i})\ ;$ (CONSG)

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

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

—

Insertion at the End of a List Segment

$$\{\mathsf{lseg}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a}, \mathsf{k}\}$$

$$\mathsf{l} := \mathbf{cons}(\mathsf{b}, \mathsf{k})\ ; \qquad \text{(CONSNOG)}$$

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

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

$$\{\mathsf{lseg}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a}\ *\ \mathsf{j}+1 \mapsto -\ *\ \mathsf{l} \mapsto \mathsf{b}, \mathsf{k}\}$$

$$[\mathsf{j}+1] := \mathsf{l} \qquad \text{(MUG)}$$

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

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

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

$$\{\mathsf{lseg}\ \alpha{\cdot}\mathsf{a}{\cdot}\mathsf{b}\ (\mathsf{i},\mathsf{k})\}.$$

—

Deletion at the Beginning of a List Segment

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

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

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

$\mathsf{j} := [\mathsf{i} + 1]\ ;$ (LKNOG)

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

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

$\mathbf{dispose}\ \mathsf{i}\ ;$ (DISG)

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

$\mathbf{dispose}\ \mathsf{i} + 1\ ;$ (DISG)

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

$\mathsf{i} := \mathsf{j}$ (AS)

$\{\mathsf{lseg}\ \alpha\ (\mathsf{i}, \mathsf{k})\}.$

Deletion at the End of a List Segment

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

$[\,\mathsf{j} + 1] := \mathsf{l}\,;$ (MUG)

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

$\textbf{dispose}\ \mathsf{k}\,;$ (DISG)

$\textbf{dispose}\ \mathsf{k} + 1$ (DISG)

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

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

—

A Cyclic Buffer

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

When $\mathsf{i} = \mathsf{j}$, the buffer is either empty ($\#\alpha = 0 \wedge \mathsf{m} = 0$) or full ($\#\beta = 0 \wedge \mathsf{m} = \mathsf{n}$).

—

Insertion in a Cyclic Buffer

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

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

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

$$[\mathsf{j}] := \mathsf{x}\ ;\ \ \}\exists\beta, \mathsf{j}'' \qquad (MUG)$$

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

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

$$\mathsf{j} := [\mathsf{j} + 1]\ ;\ \ \}\exists\beta \qquad (LKG)$$

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

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

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

$$\mathsf{m} := \mathsf{m} + 1 \qquad (AS)$$

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

—

Note the use of (LKG):

$$\frac{}{\{\exists v''.\ (e \mapsto v'')\ *\ (r/v' \to v)\}\ v := [e]\ \{\exists v'.\ (e' \mapsto v)\ *\ (r/v'' \to v)\},}$$

where v, v', and v'' are distinct, $v', v'' \notin \mathsf{FV}(e)$, $v \notin \mathsf{FV}(r)$, and e' denotes $e/v \to v'$.

with v, v', and v'' replaced by j, j', and j''; e replaced by $\mathsf{j} + 1$; and r replaced by

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

followed by (EQ) with v replaced by β, in the step

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

$$\mathsf{j} := [\mathsf{j} + 1]\ ;\ \Big\}\exists\beta \qquad (LKG)$$

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

—

A Preciseness Proof

Proposition 12 (1) $\exists\alpha.\ \mathsf{list}\ \alpha\ \mathsf{i}$ *is a precise assertion.* (2) $\mathsf{list}\ \alpha\ \mathsf{i}$ *is a precise assertion.*

PROOF (1) We begin with two preliminary properties of the list predicate:

(a) Suppose $[\,\mathsf{i}{:}\, i \mid \alpha{:}\, \epsilon\,], h \models \mathsf{list}\ \alpha\ \mathsf{i}$. Then

$$\begin{aligned}
&[\,\mathsf{i}{:}\, i \mid \alpha{:}\, \epsilon\,], h \models \mathsf{list}\ \alpha\ \mathsf{i} \wedge \alpha = \epsilon \\
&[\,\mathsf{i}{:}\, i \mid \alpha{:}\, \epsilon\,], h \models \mathsf{list}\ \epsilon\ \mathsf{i} \\
&[\,\mathsf{i}{:}\, i \mid \alpha{:}\, \epsilon\,], h \models \mathbf{emp} \wedge \mathsf{i} = \mathbf{nil},
\end{aligned}$$

so that h is the empty heap and $i = \mathbf{nil}$.

(b) On the other hand, suppose $[\,\mathsf{i}{:}\, i \mid \alpha{:}\, a{\cdot}\alpha'\,], h \models \mathsf{list}\ \alpha\ \mathsf{i}$. Then

$$\begin{aligned}
&[\,\mathsf{i}{:}\, i \mid \alpha{:}\, a{\cdot}\alpha' \mid \mathsf{a}{:}\, a \mid \alpha'{:}\, \alpha'\,], h \models \mathsf{list}\ \alpha\ \mathsf{i} \wedge \alpha = \mathsf{a}{\cdot}\alpha' \\
&[\,\mathsf{i}{:}\, i \mid \mathsf{a}{:}\, a \mid \alpha'{:}\, \alpha'\,], h \models \mathsf{list}\ (\mathsf{a}{\cdot}\alpha')\ \mathsf{i} \\
&[\,\mathsf{i}{:}\, i \mid \mathsf{a}{:}\, a \mid \alpha'{:}\, \alpha'\,], h \models \exists \mathsf{j}.\ \mathsf{i} \mapsto \mathsf{a}, \mathsf{j} \ast \mathsf{list}\ \alpha'\ \mathsf{j} \\
&\exists j.\ [\,\mathsf{i}{:}\, i \mid \mathsf{a}{:}\, a \mid \mathsf{j}{:}\, j \mid \alpha'{:}\, \alpha'\,], h \models \mathsf{i} \mapsto \mathsf{a}, \mathsf{j} \ast \mathsf{list}\ \alpha'\ \mathsf{j},
\end{aligned}$$

so that there are j and h' such that

$$i \neq \mathbf{nil} \qquad h = [\, i{:}\, a \mid i+1{:}\, j\,] \cdot h' \qquad [\mathsf{j}{:}\, j \mid \alpha'{:}\, \alpha'\,], h' \models \mathsf{list}\ \alpha'\ \mathsf{j},$$

and by the substitution theorem,

$$[\,\mathsf{i}{:}\, j \mid \alpha{:}\, \alpha'\,], h' \models \mathsf{list}\ \alpha\ \mathsf{i}.$$

—

A Preciseness Proof (continued)

To prove (1), we assume s, h, h_0, and h_1 are such that h_0, $h_1 \subseteq h$ and

$$s, h_0 \vDash \exists\alpha.\ \mathsf{list}\ \alpha\ \mathsf{i} \qquad\qquad s, h_1 \vDash \exists\alpha.\ \mathsf{list}\ \alpha\ \mathsf{i}.$$

We must show that $h_0 = h_1$.

Since i is the only free variable of the above assertion, we can assume s is $[\,\mathsf{i}\colon i\,]$ for some i. Then we can use the semantic equation for the existential quantifier to show that there are sequences α_0 and α_1 such that

$$[\,\mathsf{i}\colon i \mid \alpha\colon \alpha_0\,], h_0 \vDash \mathsf{list}\ \alpha\ \mathsf{i} \qquad\qquad [\,\mathsf{i}\colon i \mid \alpha\colon \alpha_1\,], h_1 \vDash \mathsf{list}\ \alpha\ \mathsf{i}.$$

We will complete our proof by showing, by structural induction on α_0:

> For all α_0, α_1, i, h, h_0, and h_1, if h_0, $h_1 \subseteq h$ and the statements displayed above hold, then $h_0 = h_1$.

For the base case, suppose α_0 is empty. Then by (a), h_0 is the empty heap and $i = \mathbf{nil}$.

Moreover, if α_1 were not empty, then by (b) we would have the contradiction $i \neq \mathbf{nil}$. Thus α_1 must be empty, so by (a), h_1 is the empty heap, so that $h_0 = h_1$.

—

A Preciseness Proof (continued)

For the induction step suppose $\alpha_0 = a_0{\cdot}\alpha'_0$. Then by (b), there are j_0 and h'_0 such that

$$i \neq \mathbf{nil}, \;\; h_0 = [\, i\colon a_0 \mid i{+}1\colon j_0 \,]{\cdot}h'_0, \;\; [\, \mathsf{i}\colon j_0 \mid \alpha\colon \alpha'_0 \,], h'_0 \vDash \mathsf{list}\ \alpha\ \mathsf{i}.$$

Moreover, if α_1 were empty, then by (a) we would have the contradiction $i = \mathbf{nil}$. Thus α_1 must be $a_1{\cdot}\alpha'_1$ for some a_1 and α'_1. Then by (b), there are j_1 and h'_1 such that

$$i \neq \mathbf{nil}, \;\; h_1 = [\, i\colon a_1 \mid i{+}1\colon j_1 \,]{\cdot}h'_1, \;\; [\, \mathsf{i}\colon j_1 \mid \alpha\colon \alpha'_1 \,], h'_1 \vDash \mathsf{list}\ \alpha\ \mathsf{i}.$$

Since h_0 and h_1 are both subsets of h, they must map i and $i{+}1$ into the same value. Thus $[\, i\colon a_0 \mid i{+}1\colon j_0 \,] = [\, i\colon a_1 \mid i{+}1\colon j_1 \,]$, so that $a_0 = a_1$ and $j_0 = j_1$. Then, since

$$[\, \mathsf{i}\colon j_0 \mid \alpha\colon \alpha'_0 \,], h'_0 \vDash \mathsf{list}\ \alpha\ \mathsf{i} \quad \text{and} \quad [\, \mathsf{i}\colon j_0 \mid \alpha\colon \alpha'_1 \,], h'_1 \vDash \mathsf{list}\ \alpha\ \mathsf{i},$$

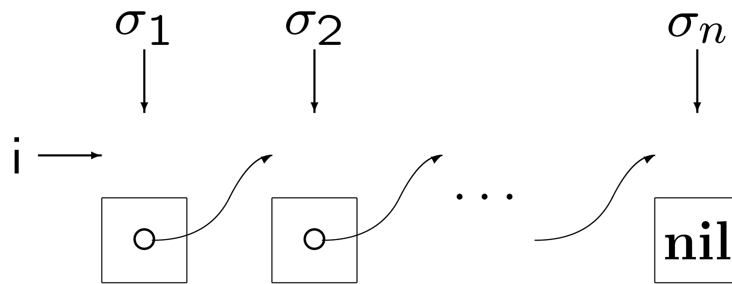
the induction hypothesis give $h'_0 = h'_1$. It follows that $h_0 = h_1$.

(2) We use the law that p is precise whenever $p \Rightarrow q$ is valid and q is precise. Then, since $\mathbf{list}\ \alpha\ \mathsf{i} \Rightarrow \exists \alpha.\ \mathbf{list}\ \alpha\ \mathsf{i}$ is valid, $\mathbf{list}\ \alpha\ \mathsf{i}$ is precise. END OF PROOF

—

Bornat Lists

listN σ i:



is defined by

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

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

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

—

Reversing a Bornat List

$$\{\mathsf{listN}\ \sigma_0\ \mathsf{i}\}$$

$$\{\mathsf{listN}\ \sigma_0\ \mathsf{i}\ *\ (\mathbf{emp} \wedge \mathbf{nil} = \mathbf{nil})\}$$

$$\mathsf{j} := \mathbf{nil}\ ;$$

$$\{\mathsf{listN}\ \sigma_0\ \mathsf{i}\ *\ (\mathbf{emp} \wedge \mathsf{j} = \mathbf{nil})\}$$

$$\{\mathsf{listN}\ \sigma_0\ \mathsf{i}\ *\ \mathsf{listN}\ \epsilon\ \mathsf{j}\}$$

$$\{\exists \sigma, \tau.\ (\mathsf{listN}\ \sigma\ \mathsf{i}\ *\ \mathsf{listN}\ \tau\ \mathsf{j}) \wedge \sigma_0^\dagger = \sigma^\dagger {\cdot} \tau\}$$

$$\mathbf{while}\ \mathsf{i} \neq \mathbf{nil}\ \mathbf{do}$$

$$\big(\{\exists \sigma, \tau.\ (\mathsf{listN}\ (\mathsf{i}{\cdot}\sigma)\ \mathsf{i}\ *\ \mathsf{listN}\ \tau\ \mathsf{j}) \wedge \sigma_0^\dagger = (\mathsf{i}{\cdot}\sigma)^\dagger {\cdot} \tau\}$$

$$\{\exists \sigma, \tau, \mathsf{k}.\ (\mathsf{i} + 1 \mapsto \mathsf{k}\ *\ \mathsf{listN}\ \sigma\ \mathsf{k}\ *\ \mathsf{listN}\ \tau\ \mathsf{j}) \\ \wedge\ \sigma_0^\dagger = (\mathsf{i}{\cdot}\sigma)^\dagger {\cdot} \tau\}$$

$$\mathsf{k} := [\mathsf{i} + 1]\ ;$$

$$\{\exists \sigma, \tau.\ (\mathsf{i} + 1 \mapsto \mathsf{k}\ *\ \mathsf{listN}\ \sigma\ \mathsf{k}\ *\ \mathsf{listN}\ \tau\ \mathsf{j}) \\ \wedge\ \sigma_0^\dagger = (\mathsf{i}{\cdot}\sigma)^\dagger {\cdot} \tau\}$$

$$[\mathsf{i} + 1] := \mathsf{j}\ ;$$

$$\{\exists \sigma, \tau.\ (\mathsf{i} + 1 \mapsto \mathsf{j}\ *\ \mathsf{listN}\ \sigma\ \mathsf{k}\ *\ \mathsf{listN}\ \tau\ \mathsf{j}) \\ \wedge\ \sigma_0^\dagger = (\mathsf{i}{\cdot}\sigma)^\dagger {\cdot} \tau\}$$

$$\{\exists \sigma, \tau.\ (\mathsf{listN}\ \sigma\ \mathsf{k}\ *\ \mathsf{listN}\ (\mathsf{i}{\cdot}\tau)\ \mathsf{i}) \wedge \sigma_0^\dagger = \sigma^\dagger {\cdot} \mathsf{i} {\cdot} \tau\}$$

$$\{\exists \sigma, \tau.\ (\mathsf{listN}\ \sigma\ \mathsf{k}\ *\ \mathsf{listN}\ \tau\ \mathsf{i}) \wedge \sigma_0^\dagger = \sigma^\dagger {\cdot} \tau\}$$

$$\mathsf{j} := \mathsf{i}\ ;\ \mathsf{i} := \mathsf{k}$$

$$\{\exists \sigma, \tau.\ (\mathsf{listN}\ \sigma\ \mathsf{i}\ *\ \mathsf{listN}\ \tau\ \mathsf{j}) \wedge \sigma_0^\dagger = \sigma^\dagger {\cdot} \tau\}\big)$$

$$\{\exists \sigma, \tau.\ \mathsf{listN}\ \tau\ \mathsf{j} \wedge \sigma_0^\dagger = \sigma^\dagger {\cdot} \tau \wedge \sigma = \epsilon\}$$

$$\{\mathsf{listN}\ \sigma_0^\dagger\ \mathsf{j}\}$$

Simple Procedures

By “simple” procedures, we mean that the following restrictions are imposed:

- Parameters are variables and expressions, not commands or procedure names.
- There are no “global” variables: All free variables of the procedure body must be formal parameters of the procedure.
- Procedures are proper, i.e., their calls are commands.
- Calls are restricted to prevent aliasing.

An additional peculiarity, which substantially simplifies reasoning about simple procedures, is that we syntactically distinguish parameters that may be modified from those that may not be.

—

Procedure Definitions

A *simple nonrecursive (or recursive) procedure definition* is a command of the form

$$\mathbf{let}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) = c\ \mathbf{in}\ c'$$
$$\mathbf{letrec}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) = c\ \mathbf{in}\ c',$$

where

- h is a binding occurrence of a procedure name, whose scope is c' (or c and c' in the recursive case).
- c and c' are commands.
- $v_1, \ldots, v_m; v'_1, \ldots, v'_n$ is a list of distinct variables, called *formal parameters*, that includes all of the free variables of c. The formal parameters are binding occurrences whose scope is c.
- $v_1, \ldots, v_m$ includes all of the variables modified by c.

—

Procedure Calls

A *procedure call* is a command of the form

$$h(w_1, \ldots, w_m; e'_1, \ldots, e'_n),$$

where

- h is a procedure name.
- $w_1, \ldots, w_m$ and $e'_1, \ldots, e'_n$ are called *actual parameters*.
- $w_1, \ldots, w_m$ are distinct variables.
- $e'_1, \ldots, e'_n$ are expressions that do not contain occurrences of the variables $w_1, \ldots, w_m$.
- The free variables of the procedure call are

  $$\begin{aligned} &\mathsf{FV}(h(w_1, \ldots, w_m; e'_1, \ldots, e'_n)) = \\ &\qquad \{w_1, \ldots, w_m\} \cup \mathsf{FV}(e'_1) \cup \cdots \cup \mathsf{FV}(e'_n) \end{aligned}$$

  and the variables modified by the call are $w_1, \ldots, w_m$.

—

Hypothetical Specifications

The truth of a specification $\{p\}\ c\ \{q\}$ will depend upon an *environment*, which maps the procedure names occurring free in c into their meanings.

We define a *hypothetical specification* to have the form

$$\Gamma \vdash \{p\}\ c\ \{q\},$$

where the *context* Γ is a sequence of specifications of the form

$$\{p_0\}\ c_0\ \{q_0\}, \ldots, \{p_{n-1}\}\ c_{n-1}\ \{q_{n-1}\}.$$

We say that such a hypothetical specification is true iff $\{p\}\ c\ \{q\}$ holds for every environment in which all of the specifications in Γ hold.

—

Generalizing Old Inference Rules

For example,

- Strengthening Precedent (SP)

$$\frac{p \Rightarrow q \qquad \Gamma \vdash \{q\}\ c\ \{r\}}{\Gamma \vdash \{p\}\ c\ \{r\}.}$$

- Substitution (SUB)

$$\frac{\Gamma \vdash \{p\}\ c\ \{q\}}{\Gamma \vdash \{p/\delta\}\ (c/\delta)\ \{q/\delta\},}$$

  where δ is the substitution $v_1 \rightarrow e_1, \ldots, v_n \rightarrow e_n$; $v_1, \ldots, v_n$ are the variables occurring free in p, c, or q; and, if v_i is modified by c, then e_i is a variable that does not occur free in any other e_j.

Note that substitutions do not affect procedure names.

—

Rules for Procedures

- Hypothesis (HYPO)

$$\frac{}{\Gamma, \{p\}\ c\ \{q\}, \Gamma' \vdash \{p\}\ c\ \{q\}.}$$

- Simple Procedures (SPROC)

$$\frac{\Gamma \vdash \{p\}\ c\ \{q\} \qquad \Gamma, \{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\ \{q\} \vdash \{p'\}\ c'\ \{q'\}}{\Gamma \vdash \{p'\}\ \mathbf{let}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) = c\ \mathbf{in}\ c'\ \{q'\},}$$

where h does not occur free in any triple of Γ.

- Simple Recursive Procedures (SRPROC)
  (partial correctness only)

$$\frac{\Gamma, \{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\ \{q\} \vdash \{p\}\ c\ \{q\} \qquad \Gamma, \{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\ \{q\} \vdash \{p'\}\ c'\ \{q'\}}{\Gamma \vdash \{p'\}\ \mathbf{letrec}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) = c\ \mathbf{in}\ c'\ \{q'\},}$$

where h does not occur free in any triple of Γ.

—

Some Limitations

To keep our exposition straightforward, we have ignored:

- Simultaneous recursion,
- Multiple hypotheses for the same procedure.

—

Two Derived Rules

From (HYPO):

- Call (CALL)

$$\frac{}{\Gamma, \{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\ \{q\}, \Gamma' \vdash \{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\ \{q\}.}$$

and from (CALL) and (SUB):

- General Call (GCALL)

$$\frac{}{\Gamma, \{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\ \{q\}, \Gamma' \vdash \{p/\delta\}\ h(w_1, \ldots, w_m; e'_1, \ldots, e'_n)\ \{q/\delta\},}$$

  where δ is a substitution

$$\begin{aligned}\delta = v_1 \to w_1, \ldots, v_m \to w_m,\\ v'_1 \to e'_1, \ldots, v'_n \to e'_n,\\ v''_1 \to e''_1, \ldots, v''_k \to e''_k,\end{aligned}$$

  which acts on all the free variables in

$$\{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\ \{q\},$$

  and $w_1, \ldots, w_m$ are distinct variables that do not occur free in the expressions $e'_1, \ldots, e'_n$ or $e''_1, \ldots, e''_k$.

—

Annotated Specifications: Ghosts

In (GCALL):

$$\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$$

$$\Gamma, \{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\ \{q\}, \Gamma' \vdash$$
$$\{p/\delta\}\ h(w_1, \ldots, w_m; e'_1, \ldots, e'_n)\ \{q/\delta\},$$

where δ is a substitution

$$\begin{aligned}\delta = v_1 \to w_1, \ldots, v_m \to w_m,\\ v'_1 \to e'_1, \ldots, v'_n \to e'_n,\\ v''_1 \to e''_1, \ldots, v''_k \to e''_k,\end{aligned}$$

which acts on . . . .

there may be ghost variables $v''_1, \ldots, v''_k$ that appear in δ but are not formal parameters.

We will treat $v''_1, \ldots, v''_k$ as formal ghost parameters, and $e''_1, \ldots, e''_k$ as actual ghost parameters.

—

For example,

$$\left.\begin{array}{l}\{\mathsf{n} \geq 0 \wedge \mathsf{r} = \mathsf{r}_0\} \\ \mathsf{multfact}(\mathsf{r};\mathsf{n}) \\ \{\mathsf{r} = \mathsf{n}! \times \mathsf{r}_0\}\end{array}\right\} \vdash \left\{\begin{array}{l}\{\mathsf{n}-1 \geq 0 \wedge \mathsf{r} = \mathsf{n} \times \mathsf{r}_0\} \\ \mathsf{multfact}(\mathsf{r};\mathsf{n}-1) \\ \{\mathsf{r} = (\mathsf{n}-1)! \times \mathsf{n} \times \mathsf{r}_0\}\end{array}\right.$$

is an instance of (GCALL) using the substitution

$$\mathsf{r} \to \mathsf{r}, \mathsf{n} \to \mathsf{n}-1, \mathsf{r}_0 \to \mathsf{n} \times \mathsf{r}_0.$$

The corresponding annotated specification will be

$$\left.\begin{array}{l}\{\mathsf{n} \geq 0 \wedge \mathsf{r} = \mathsf{r}_0\} \\ \mathsf{multfact}(\mathsf{r};\mathsf{n})\underline{\{\mathsf{r}_0\}} \\ \{\mathsf{r} = \mathsf{n}! \times \mathsf{r}_0\}\end{array}\right\} \vdash \left\{\begin{array}{l}\{\mathsf{n}-1 \geq 0 \wedge \mathsf{r} = \mathsf{n} \times \mathsf{r}_0\} \\ \mathsf{multfact}(\mathsf{r};\mathsf{n}-1)\underline{\{\mathsf{n} \times \mathsf{r}_0\}} \\ \{\mathsf{r} = (\mathsf{n}-1)! \times \mathsf{n} \times \mathsf{r}_0\}.\end{array}\right.$$

—

Generalizing Annotation Descriptions

An *annotated context* is a sequence of *annotated hypotheses*, which have the form

$$\{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\{v''_1, \ldots, v''_k\}\ \{q\},$$

where $v''_1, \ldots, v''_k$ is a list of formal ghost parameters (and all of the formal parameters, including the ghosts, are distinct).

We write $\widehat{\Gamma}$ to denote an annotated context, and Γ to denote the corresponding ordinary context that is obtained by erasing the lists of ghost formal parameters. Then an annotation description has the form:

$$\widehat{\Gamma} \vdash \mathcal{A} \gg \{p\}\ c\ \{q\},$$

meaning that $\widehat{\Gamma} \vdash \mathcal{A}$ is an annotated hypothetical specification proving the hypothetical specification $\Gamma \vdash \{p\}\ c\ \{q\}$.

—

Rules for Procedural Annotated Specifications

- General Call (GCALLan)

$$\hat{\Gamma}, \{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\{v''_1, \ldots, v''_k\}\ \{q\}, \hat{\Gamma}' \vdash$$
$$h(w_1, \ldots, w_m; e'_1, \ldots, e'_n)\{e''_1, \ldots, e''_k\} \gg$$
$$\{p/\delta\}\ h(w_1, \ldots, w_m; e'_1, \ldots, e'_n)\ \{q/\delta\},$$

where δ is the substitution

$$\begin{aligned}\delta = v_1 \rightarrow w_1, \ldots, v_m \rightarrow w_m,\\ v'_1 \rightarrow e'_1, \ldots, v'_n \rightarrow e'_n,\\ v''_1 \rightarrow e''_1, \ldots, v''_k \rightarrow e''_k,\end{aligned}$$

which acts on all the free variables in

$$\{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\ \{q\},$$

and $w_1, \ldots, w_m$ are distinct variables that do not occur free in the expressions $e'_1, \ldots, e'_n$ or $e''_1, \ldots, e''_k$.

—

- Simple Procedures (SPROCan)

$$\frac{\widehat{\Gamma} \vdash \mathcal{A} \gg \{p\}\ c\ \{q\} \qquad \widehat{\Gamma}, \{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\{v''_1, \ldots, v''_k\}\ \{q\} \vdash \mathcal{A}' \gg \{p'\}\ c'\ \{q'\}}{\widehat{\Gamma} \vdash \mathbf{let}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\{v''_1, \ldots, v''_k\} = \mathcal{A}\ \mathbf{in}\ \mathcal{A}' \gg \{p'\}\ \mathbf{let}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) = c\ \mathbf{in}\ c'\ \{q'\}},$$

where h does not occur free in any triple of $\widehat{\Gamma}$.

—

- Simple Recursive Procedures (SRPROCan) (corrected)

$$\frac{\begin{array}{r}\widehat{\Gamma}, \{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\{v''_1, \ldots, v''_k\}\ \{q\} \vdash \\ \{p\}\ \mathcal{A}\ \{q\} \gg \{p\}\ c\ \{q\} \\ \widehat{\Gamma}, \{p\}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\{v''_1, \ldots, v''_k\}\ \{q\} \vdash \\ \mathcal{A}' \gg \{p'\}\ c'\ \{q'\}\end{array}}{\begin{array}{r}\widehat{\Gamma} \vdash \mathbf{letrec}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\{v''_1, \ldots, v''_k\} = \{p\}\ \mathcal{A}\ \{q\} \\ \mathbf{in}\ \mathcal{A}'\end{array}}$$

$$\gg \{p'\}\ \mathbf{letrec}\ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) = c\ \mathbf{in}\ c'\ \{q'\},$$

where h does not occur free in any triple of $\widehat{\Gamma}$.

—

An Example

$$
\begin{array}{l}
\{\mathsf{z} = 10\} \\
\textbf{letrec}\ \mathsf{multfact}(\mathsf{r};\mathsf{n})\{\mathsf{r}_0\} = \\
\quad \{\mathsf{n} \geq 0 \wedge \mathsf{r} = \mathsf{r}_0\} \\
\quad \textbf{if}\ \mathsf{n} = 0\ \textbf{then} \\
\qquad \{\mathsf{n} = 0 \wedge \mathsf{r} = \mathsf{r}_0\}\ \textbf{skip}\ \{\mathsf{r} = \mathsf{n}! \times \mathsf{r}_0\} \\
\quad \textbf{else} \\
\qquad \{\mathsf{n} - 1 \geq 0 \wedge \mathsf{n} \times \mathsf{r} = \mathsf{n} \times \mathsf{r}_0\} \\
\qquad \mathsf{r} := \mathsf{n} \times \mathsf{r}\ ; \\
\qquad \{\mathsf{n} - 1 \geq 0 \wedge \mathsf{r} = \mathsf{n} \times \mathsf{r}_0\} \\
\qquad \left.\begin{array}{l}
\quad \{\mathsf{n} - 1 \geq 0 \wedge \mathsf{r} = \mathsf{n} \times \mathsf{r}_0\} \\
\quad \mathsf{multfact}(\mathsf{r};\mathsf{n} - 1)\{\mathsf{n} \times \mathsf{r}_0\} \\
\quad \{\mathsf{r} = (\mathsf{n} - 1)! \times \mathsf{n} \times \mathsf{r}_0\}
\end{array}\right\} * \ \mathsf{n} - 1 \geq 0 \quad \begin{array}{c}(*) \\ \\ (*)\end{array} \\
\qquad \{\mathsf{n} - 1 \geq 0 \wedge \mathsf{r} = (\mathsf{n} - 1)! \times \mathsf{n} \times \mathsf{r}_0\} \\
\quad \{\mathsf{r} = \mathsf{n}! \times \mathsf{r}_0\} \\
\textbf{in} \\
\{5 \geq 0 \wedge \mathsf{z} = 10\} \qquad (*) \\
\mathsf{multfact}(\mathsf{z};5)\{10\} \\
\{\mathsf{z} = 5! \times 10\} \qquad (*)
\end{array}
$$

—

An Example (continued)

$\{z = 10\}$

$\textbf{letrec}\ \mathsf{multfact}(r; n)\{r_0\} =$

$\quad \{n \geq 0 \wedge r = r_0\}$

$\quad\quad \vdots$

$\quad \{r = n! \times r_0\}$

$\textbf{in}$

$\{5 \geq 0 \wedge z = 10\}$ $\quad (*)$

$\mathsf{multfact}(z; 5)\{10\}$

$\{z = 5! \times 10\}$ $\quad (*)$

—

How the Annotations Determine a Formal Proof

The application of (SRPROCan) to the **letrec** definition gives rise to the hypothesis

$$\{\mathsf{n} \geq 0 \wedge \mathsf{r} = \mathsf{r}_0\}\ \mathsf{multfact}(\mathsf{r};\mathsf{n})\{\mathsf{r}_0\}\ \{\mathsf{r} = \mathsf{n}! \times \mathsf{r}_0\}.$$

By (GCALLan), the hypothesis entails

$$\begin{array}{l} \{\mathsf{n}-1 \geq 0 \wedge \mathsf{r} = \mathsf{n} \times \mathsf{r}_0\} \\ \mathsf{multfact}(\mathsf{r};\mathsf{n}-1)\{\mathsf{n} \times \mathsf{r}_0\} \\ \{\mathsf{r} = (\mathsf{n}-1)! \times \mathsf{n} \times \mathsf{r}_0\}. \end{array}$$

Next, since n is not modified by the call $\mathsf{multfact}(\mathsf{r};\mathsf{n}-1)$, the frame rule gives

$$\begin{array}{l} \{\mathsf{n}-1 \geq 0 \wedge \mathsf{r} = \mathsf{n} \times \mathsf{r}_0\ *\ \mathsf{n}-1 \geq 0\} \\ \mathsf{multfact}(\mathsf{r};\mathsf{n}-1)\{\mathsf{n} \times \mathsf{r}_0\} \\ \{\mathsf{r} = (\mathsf{n}-1)! \times \mathsf{n} \times \mathsf{r}_0\ *\ \mathsf{n}-1 \geq 0\}. \end{array}$$

But the assertions here are all pure, so that the separating conjunctions can be replaced by ordinary conjunctions. Then, we can strengthen the precondition and weaken the postcondition, to obtain

$$\begin{array}{l} \{\mathsf{n}-1 \geq 0 \wedge \mathsf{r} = \mathsf{n} \times \mathsf{r}_0\} \\ \mathsf{multfact}(\mathsf{r};\mathsf{n}-1)\{\mathsf{n} \times \mathsf{r}_0\} \\ \{\mathsf{n}-1 \geq 0 \wedge \mathsf{r} = (\mathsf{n}-1)! \times \mathsf{n} \times \mathsf{r}_0\}. \end{array}$$

Also, by (GCALLan), the hypothesis entails

$$\{5 \geq 0 \wedge \mathsf{z} = 10\}\ \mathsf{multfact}(\mathsf{z};5)\{10\}\ \{\mathsf{z} = 5! \times 10\}.$$

—

Some Concepts about Sequences: Images

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

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

of values occurring as components of α. It satisfies the laws:

$$\{\epsilon\} = \{\} \tag{1}$$

$$\{[x]\} = \{x\} \tag{2}$$

$$\{\alpha \cdot \beta\} = \{\alpha\} \cup \{\beta\} \tag{3}$$

$$\#\{\alpha\} \leq \#\alpha. \tag{4}$$

—

Pointwise Extension of Binary Relations

If ρ is a relation between values, then ρ^* is the 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 the laws:

$$S' \subseteq S \wedge S\ \rho^*\ T \;\Rightarrow\; S'\ \rho^*\ T \tag{5}$$

$$T' \subseteq T \wedge S\ \rho^*\ T \;\Rightarrow\; S\ \rho^*\ T' \tag{6}$$

$$\{\}\ \ \rho^*\ \ T \tag{7}$$

$$S\ \ \rho^*\ \ \{\} \tag{8}$$

$$\{x\}\ \rho^*\ \{y\} \;\Leftrightarrow\; x\ \rho\ y \tag{9}$$

$$(S \cup S')\ \rho^*\ T \;\Leftrightarrow\; S\ \rho^*\ T \wedge S'\ \rho^*\ T \tag{10}$$

$$S\ \rho^*\ (T \cup T') \;\Leftrightarrow\; S\ \rho^*\ T \wedge S\ \rho^*\ T'. \tag{11}$$

The following abbreviations are also useful:

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

—

Ordering

We write $\mathbf{ord}\ \alpha$ if the sequence α is ordered in nonstrict increasing order. Then $\mathbf{ord}$ satisfies

$$\#\alpha \leq 1 \;\Rightarrow\; \mathbf{ord}\ \alpha \tag{12}$$

$$\mathbf{ord}\ \alpha{\cdot}\beta \;\Leftrightarrow\; \mathbf{ord}\ \alpha \wedge \mathbf{ord}\ \beta \wedge \{\alpha\} \leq^* \{\beta\} \tag{13}$$

$$\mathbf{ord}\ [x]{\cdot}\alpha \;\Rightarrow\; x \leq^* \{[x]{\cdot}\alpha\} \tag{14}$$

$$\mathbf{ord}\ \alpha{\cdot}[x] \;\Rightarrow\; \{\alpha{\cdot}[x]\} \leq^* x. \tag{15}$$

—

Rearrangement

We say that a sequence β is a *rearrangement* of a sequence α, written $\beta \sim \alpha$, iff there is a permutation ϕ, from the domain $(1 \text{ to } \#\beta)$ of β to the domain $(1 \text{ to } \#\alpha)$ of α, such that

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

Then

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

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

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

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

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

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

$$\alpha \sim \beta \;\Rightarrow\; \#\alpha = \#\beta. \quad (22)$$

—

Sorting by Merging: Lists with Explicit Lengths

The basic idea behind sorting by merging is to divide the input list segment into two roughly equal halves, sort each half recursively, and then merge the results. Unfortunately, however, one cannot divide a list segment into two halves efficiently.

A way around this difficulty is to give the lengths of the input segments to the commands for sorting and merging as explicit numbers.

We define

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

—

Then we will define a procedure mergesort satisfying the hypothesis

$$
\begin{aligned}
H_{\mathsf{mergesort}} \stackrel{\text{def}}{=} \ & \{\mathsf{lseg}\ \alpha\,(\mathsf{i}, \mathsf{j}_0) \wedge \#\alpha = \mathsf{n} \wedge \mathsf{n} \geq 1\} \\
& \mathsf{mergesort}(\mathsf{i}, \mathsf{j}; \mathsf{n})\{\alpha, \mathsf{j}_0\} \\
& \{\exists\beta.\ \mathsf{lseg}\ \beta\,(\mathsf{i}, -) \wedge \beta \sim \alpha \wedge \mathbf{ord}\ \beta \wedge \mathsf{j} = \mathsf{j}_0\}.
\end{aligned}
$$

The subsidiary procedure merge will satisfy

$$
\begin{aligned}
H_{\mathsf{merge}} \stackrel{\text{def}}{=} \ & \{(\mathsf{lseg}\ \beta_1\,(\mathsf{i1}, -) \wedge \mathbf{ord}\ \beta_1 \wedge \#\beta_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1) \\
& \ \ * (\mathsf{lseg}\ \beta_2\,(\mathsf{i2}, -) \wedge \mathbf{ord}\ \beta_2 \wedge \#\beta_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1)\} \\
& \mathsf{merge}(\mathsf{i}; \mathsf{n1}, \mathsf{n2}, \mathsf{i1}, \mathsf{i2})\{\beta_1, \beta_2\} \\
& \{\exists\beta.\ \mathsf{lseg}\ \beta\,(\mathsf{i}, -) \wedge \beta \sim \beta_1{\cdot}\beta_2 \wedge \mathbf{ord}\ \beta\}.
\end{aligned}
$$

—

A Proof for mergesort

$$
\begin{aligned}
H_{\mathsf{mergesort}}, H_{\mathsf{merge}} \vdash \quad & \{\mathsf{lseg}\ \alpha\ (\mathsf{i}, \mathsf{j}_0) \wedge \#\alpha = \mathsf{n} \wedge \mathsf{n} \geq 1\} \\
& \mathbf{if}\ \mathsf{n} = 1\ \mathbf{then} \\
& \quad \{\mathsf{lseg}\ \alpha\ (\mathsf{i}, -) \wedge \mathbf{ord}\ \alpha \wedge \mathsf{i} \mapsto -, \mathsf{j}_0\} \\
& \quad \mathsf{j} := [\mathsf{i} + 1] \\
& \quad \{\mathsf{lseg}\ \alpha\ (\mathsf{i}, -) \wedge \mathbf{ord}\ \alpha \wedge \mathsf{j} = \mathsf{j}_0\} \\
& \mathbf{else} \\
& \quad \vdots
\end{aligned}
$$

—

$\vdots$

$\textbf{else newvar } \mathsf{n1} \textbf{ in newvar } \mathsf{n2} \textbf{ in newvar } \mathsf{i1} \textbf{ in newvar } \mathsf{i2} \textbf{ in}$

$\big(\mathsf{n1} := \mathsf{n} \div 2\ ;\ \mathsf{n2} := \mathsf{n} - \mathsf{n1}\ ;\ \mathsf{i1} := \mathsf{i}\ ;$

$$\begin{array}{l}
\{\exists \alpha_1, \alpha_2, \mathsf{i}_2.\ (\mathsf{lseg}\ \alpha_1\ (\mathsf{i1}, \mathsf{i}_2)\ *\ \mathsf{lseg}\ \alpha_2\ (\mathsf{i}_2, \mathsf{j}_0)) \\
\quad \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2\}
\end{array}$$

$$\left.\begin{array}{l}
\left.\begin{array}{l}
\{\mathsf{lseg}\ \alpha_1\ (\mathsf{i1}, \mathsf{i}_2) \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1\} \\
\mathsf{mergesort}(\mathsf{i1}, \mathsf{i2}; \mathsf{n1})\{\alpha_1, \mathsf{i}_2\}\ ; \\
\{\exists \beta.\ \mathsf{lseg}\ \beta\ (\mathsf{i1}, -) \wedge \beta{\sim}\alpha_1 \wedge \mathbf{ord}\ \beta \wedge \mathsf{i2} = \mathsf{i}_2\} \\
\{\exists \beta_1.\ \mathsf{lseg}\ \beta_1\ (\mathsf{i1}, -) \wedge \beta_1{\sim}\alpha_1 \wedge \mathbf{ord}\ \beta_1 \wedge \mathsf{i2} = \mathsf{i}_2\}
\end{array}\right\} \\
\qquad *\ (\mathsf{lseg}\ \alpha_2(\mathsf{i}_2, \mathsf{j}_0) \wedge \#\alpha_1 = \mathsf{n1} \\
\qquad\quad \wedge\ \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2)
\end{array}\right\} \exists \alpha_1, \alpha_2, \mathsf{i}_2$$

$$\begin{array}{l}
\{\exists \alpha_1, \alpha_2, \beta_1. \\
\quad (\mathsf{lseg}\ \beta_1\ (\mathsf{i1}, -)\ *\ \mathsf{lseg}\ \alpha_2\ (\mathsf{i2}, \mathsf{j}_0)) \wedge \beta_1{\sim}\alpha_1 \wedge \mathbf{ord}\ \beta_1 \\
\quad \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2\}
\end{array}$$

$$\left.\begin{array}{l}
\left.\begin{array}{l}
\{\mathsf{lseg}\ \alpha_2\ (\mathsf{i2}, \mathsf{j}_0) \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1\} \\
\mathsf{mergesort}(\mathsf{i2}, \mathsf{j}; \mathsf{n2})\{\alpha_2, \mathsf{j}_0\}\ ; \\
\{\exists \beta.\ \mathsf{lseg}\ \beta\ (\mathsf{i2}, -) \wedge \beta{\sim}\alpha_2 \wedge \mathbf{ord}\ \beta \wedge \mathsf{j} = \mathsf{j}_0\} \\
\{\exists \beta_2.\ \mathsf{lseg}\ \beta_2\ (\mathsf{i2}, -) \wedge \beta_2{\sim}\alpha_2 \wedge \mathbf{ord}\ \beta_2 \wedge \mathsf{j} = \mathsf{j}_0\}
\end{array}\right\} \\
\qquad *\, (\mathsf{lseg}\ \beta_1(\mathsf{i1}, -) \wedge \beta_1{\sim}\alpha_1 \wedge \mathbf{ord}\ \beta_1 \wedge \#\alpha_1 = \mathsf{n1} \\
\qquad\quad \wedge\ \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2)
\end{array}\right\} \exists \alpha_1, \alpha_2, \beta_1$$

$$\begin{array}{l}
\{\exists \alpha_1, \alpha_2, \beta_1, \beta_2. \\
\quad ((\mathsf{lseg}\ \beta_1\ (\mathsf{i1}, -) \wedge \beta_1{\sim}\alpha_1 \wedge \mathbf{ord}\ \beta_1 \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1) \\
\quad *\ (\mathsf{lseg}\ \beta_2\ (\mathsf{i2}, -) \wedge \beta_2{\sim}\alpha_2 \wedge \mathbf{ord}\ \beta_2 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1)) \\
\quad \wedge\ \alpha = \alpha_1{\cdot}\alpha_2 \wedge \mathsf{j} = \mathsf{j}_0\}
\end{array}$$

$\vdots$

—

$\vdots$

$$
\begin{array}{l}
\{\exists \alpha_1, \alpha_2, \beta_1, \beta_2. \\
\quad ((\mathsf{lseg}\ \beta_1\ (\mathsf{i1}, -) \wedge \beta_1 {\sim} \alpha_1 \wedge \mathbf{ord}\ \beta_1 \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1) \\
\quad\ * (\mathsf{lseg}\ \beta_2\ (\mathsf{i2}, -) \wedge \beta_2 {\sim} \alpha_2 \wedge \mathbf{ord}\ \beta_2 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1)) \\
\qquad \wedge\ \alpha = \alpha_1 \cdot \alpha_2 \wedge \mathsf{j} = \mathsf{j}_0\} \\
\{\exists \beta_1, \beta_2.\ ((\mathsf{lseg}\ \beta_1\ (\mathsf{i1}, -) \wedge \mathbf{ord}\ \beta_1 \wedge \#\beta_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1) \\
\quad\ * (\mathsf{lseg}\ \beta_2\ (\mathsf{i2}, -) \wedge \mathbf{ord}\ \beta_2 \wedge \#\beta_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1)) \\
\qquad \wedge\ \alpha {\sim} \beta_1 \cdot \beta_2 \wedge \mathsf{j} = \mathsf{j}_0\} \\
\{(\mathsf{lseg}\ \beta_1\ (\mathsf{i1}, -) \wedge \mathbf{ord}\ \beta_1 \wedge \#\beta_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1) \\
\quad\ *\ (\mathsf{lseg}\ \beta_2\ (\mathsf{i2}, -) \wedge \mathbf{ord}\ \beta_2 \wedge \#\beta_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1)\} \\
\mathsf{merge}(\mathsf{i}; \mathsf{n1}, \mathsf{n2}, \mathsf{i1}, \mathsf{i2})\{\beta_1, \beta_2\} \\
\{\exists \beta.\ \mathsf{lseg}\ \beta\ (\mathsf{i}, -) \wedge \beta {\sim} \beta_1 \cdot \beta_2 \wedge \mathbf{ord}\ \beta\} \\
\qquad\qquad\qquad\qquad\quad *\ (\mathbf{emp} \wedge \alpha {\sim} \beta_1 \cdot \beta_2 \wedge \mathsf{j} = \mathsf{j}_0) \qquad \exists \beta_1, \beta_2 \\
\{\exists \beta_1, \beta_2, \beta.\ \mathsf{lseg}\ \beta(\mathsf{i}, -) \wedge \beta {\sim} \beta_1 \cdot \beta_2 \wedge \mathbf{ord}\ \beta \\
\quad \wedge\ \alpha {\sim} \beta_1 \cdot \beta_2 \wedge \mathsf{j} = \mathsf{j}_0\}\Big) \\
\{\exists \beta.\ \mathsf{lseg}\ \beta\ (\mathsf{i}, -) \wedge \beta {\sim} \alpha \wedge \mathbf{ord}\ \beta \wedge \mathsf{j} = \mathsf{j}_0\}.
\end{array}
$$

—

An Arithmetic Subtlety

In the else branch of mergesort, to determine the division of the input list segment, the variables n1 and n2 must be set to two positive integers whose sum is n.

At this point, the length n of the input list segment is at least two. Then $2 \leq n$ and $0 \leq n - 2$, so that $2 \leq n \leq 2 \times n - 2$, 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 $n1 = n \div 2$ and $n2 = n - n1$, we have

$$1 \leq n1 \leq n - 1 \qquad 1 \leq n2 \leq n - 1 \qquad n1 + n2 = n.$$

—

Reasoning about the First Call of mergesort

$$\left.\begin{array}{l}\left.\begin{array}{l}\{\mathsf{lseg}\ \alpha_1\ (\mathsf{i1}, \mathsf{i}_2) \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1\} \\ \mathsf{mergesort}(\mathsf{i1}, \mathsf{i2}; \mathsf{n1})\{\alpha_1, \mathsf{i}_2\}\ ; \\ \{\exists\beta.\ \mathsf{lseg}\ \beta\ (\mathsf{i1}, -) \wedge \beta{\sim}\alpha_1 \wedge \mathbf{ord}\ \beta \wedge \mathsf{i2} = \mathsf{i}_2\} \\ \{\exists\beta_1.\ \mathsf{lseg}\ \beta_1\ (\mathsf{i1}, -) \wedge \beta_1{\sim}\alpha_1 \wedge \mathbf{ord}\ \beta_1 \wedge \mathsf{i2} = \mathsf{i}_2\}\end{array}\right\} \\ \quad * \ (\mathsf{lseg}\ \alpha_2(\mathsf{i}_2, \mathsf{j}_0) \wedge \#\alpha_1 = \mathsf{n1} \\ \quad\ \wedge\ \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2)\end{array}\right\} \exists\alpha_1, \alpha_2, \mathsf{i}_2$$

From the hypothesis

$$H_{\mathsf{mergesort}} \stackrel{\mathsf{def}}{=} \begin{array}{l}\{\mathsf{lseg}\ \alpha\ (\mathsf{i}, \mathsf{j}_0) \wedge \#\alpha = \mathsf{n} \wedge \mathsf{n} \geq 1\} \\ \mathsf{mergesort}(\mathsf{i}, \mathsf{j}; \mathsf{n})\{\alpha, \mathsf{j}_0\} \\ \{\exists\beta.\ \mathsf{lseg}\ \beta\ (\mathsf{i}, -) \wedge \beta \sim \alpha \wedge \mathbf{ord}\ \beta \wedge \mathsf{j} = \mathsf{j}_0\},\end{array}$$

(GCALL) is used to infer

$$\begin{array}{l}\{\mathsf{lseg}\ \alpha_1\ (\mathsf{i1}, \mathsf{i}_2) \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1\} \\ \mathsf{mergesort}(\mathsf{i1}, \mathsf{i2}; \mathsf{n1})\{\alpha_1, \mathsf{i}_2\} \\ \{\exists\beta.\ \mathsf{lseg}\ \beta\ (\mathsf{i1}, -) \wedge \beta \sim \alpha_1 \wedge \mathbf{ord}\ \beta \wedge \mathsf{i2} = \mathsf{i}_2\}\}.\end{array}$$

—

$$\left.\begin{array}{l}\left.\begin{array}{l}\{\mathsf{lseg}\ \alpha_1\ (\mathsf{i1}, \mathsf{i}_2) \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1\} \\ \mathsf{mergesort}(\mathsf{i1}, \mathsf{i2}; \mathsf{n1})\{\alpha_1, \mathsf{i}_2\}\ ; \\ \{\exists\beta.\ \mathsf{lseg}\ \beta\ (\mathsf{i1}, -) \wedge \beta{\sim}\alpha_1 \wedge \mathbf{ord}\ \beta \wedge \mathsf{i2} = \mathsf{i}_2\} \\ \{\exists\beta_1.\ \mathsf{lseg}\ \beta_1\ (\mathsf{i1}, -) \wedge \beta_1{\sim}\alpha_1 \wedge \mathbf{ord}\ \beta_1 \wedge \mathsf{i2} = \mathsf{i}_2\}\end{array}\right\} \\ \quad * \ (\mathsf{lseg}\ \alpha_2(\mathsf{i}_2, \mathsf{j}_0) \wedge \#\alpha_1 = \mathsf{n1} \\ \qquad \wedge\ \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2)\end{array}\right\} \exists \alpha_1, \alpha_2, \mathsf{i}_2$$

Then β is renamed β_1 in the postcondition:

$$\begin{array}{l}\{\mathsf{lseg}\ \alpha_1\ (\mathsf{i1}, \mathsf{i}_2) \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1\} \\ \mathsf{mergesort}(\mathsf{i1}, \mathsf{i2}; \mathsf{n1})\{\alpha_1, \mathsf{i}_2\} \\ \{\exists\beta_1.\ \mathsf{lseg}\ \beta_1\ (\mathsf{i1}, -) \wedge \beta_1 \sim \alpha_1 \wedge \mathbf{ord}\ \beta_1 \wedge \mathsf{i2} = \mathsf{i}_2\}.\end{array}$$

—

$$\left.\begin{array}{l}\left.\begin{array}{l}\{\mathsf{lseg}\ \alpha_1\,(\mathsf{i1},\mathsf{i}_2) \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1\} \\ \mathsf{mergesort}(\mathsf{i1},\mathsf{i2};\mathsf{n1})\{\alpha_1,\mathsf{i}_2\}\ ; \\ \{\exists\beta.\ \mathsf{lseg}\ \beta\,(\mathsf{i1},-) \wedge \beta{\sim}\alpha_1 \wedge \mathbf{ord}\,\beta \wedge \mathsf{i2} = \mathsf{i}_2\} \\ \{\exists\beta_1.\ \mathsf{lseg}\ \beta_1\,(\mathsf{i1},-) \wedge \beta_1{\sim}\alpha_1 \wedge \mathbf{ord}\,\beta_1 \wedge \mathsf{i2} = \mathsf{i}_2\}\end{array}\right\} \\ \quad * \ (\mathsf{lseg}\ \alpha_2(\mathsf{i}_2,\mathsf{j}_0) \wedge \#\alpha_1 = \mathsf{n1} \\ \qquad \wedge\ \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2)\end{array}\right\}\exists\alpha_1,\alpha_2,\mathsf{i}_2$$

Next, the frame rule is used to infer

$$\begin{array}{l}\{(\mathsf{lseg}\ \alpha_1\,(\mathsf{i1},\mathsf{i}_2) \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1) \\ \quad *\ \ (\mathsf{lseg}\ \alpha_2(\mathsf{i}_2,\mathsf{j}_0) \\ \quad \wedge\ \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2)\} \\ \mathsf{mergesort}(\mathsf{i1},\mathsf{i2};\mathsf{n1})\{\alpha_1,\mathsf{i}_2\} \\ \{(\exists\beta_1.\ \mathsf{lseg}\ \beta_1\,(\mathsf{i1},-) \wedge \beta_1 \sim \alpha_1 \wedge \mathbf{ord}\,\beta_1 \wedge \mathsf{i2} = \mathsf{i}_2) \\ \quad *\ \ (\mathsf{lseg}\ \alpha_2(\mathsf{i}_2,\mathsf{j}_0) \\ \quad \wedge\ \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2)\}.\end{array}$$

—

$$\left.\begin{array}{l}\left.\begin{array}{l}\{\mathsf{lseg}\ \alpha_1\,(\mathsf{i1},\mathsf{i}_2) \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1\} \\ \mathsf{mergesort}(\mathsf{i1},\mathsf{i2};\mathsf{n1})\{\alpha_1,\mathsf{i}_2\}\ ; \\ \{\exists\beta.\ \mathsf{lseg}\ \beta\,(\mathsf{i1},-) \wedge \beta{\sim}\alpha_1 \wedge \mathbf{ord}\,\beta \wedge \mathsf{i2} = \mathsf{i}_2\} \end{array}\right\} \\ \{\exists\beta_1.\ \mathsf{lseg}\ \beta_1\,(\mathsf{i1},-) \wedge \beta_1{\sim}\alpha_1 \wedge \mathbf{ord}\,\beta_1 \wedge \mathsf{i2} = \mathsf{i}_2\} \\ \quad * \ (\mathsf{lseg}\ \alpha_2(\mathsf{i}_2,\mathsf{j}_0) \wedge \#\alpha_1 = \mathsf{n1} \\ \quad\quad \wedge\ \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2) \end{array}\right\} \exists\alpha_1,\alpha_2,\mathsf{i}_2$$

Then the rule (EQ) for existential quantification gives

$$\begin{array}{l}\{\exists\alpha_1,\alpha_2,\mathsf{i}_2.\ (\mathsf{lseg}\ \alpha_1\,(\mathsf{i1},\mathsf{i}_2) \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1) \\ \quad * \ (\mathsf{lseg}\ \alpha_2(\mathsf{i}_2,\mathsf{j}_0) \\ \quad \wedge\ \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2)\} \\ \mathsf{mergesort}(\mathsf{i1},\mathsf{i2};\mathsf{n1})\{\alpha_1,\mathsf{i}_2\} \\ \{\exists\alpha_1,\alpha_2,\mathsf{i}_2.\ (\exists\beta_1.\ \mathsf{lseg}\ \beta_1\,(\mathsf{i1},-) \wedge \beta_1{\sim}\alpha_1 \wedge \mathbf{ord}\,\beta_1 \wedge \mathsf{i2} = \mathsf{i}_2) \\ \quad * \ (\mathsf{lseg}\ \alpha_2(\mathsf{i}_2,\mathsf{j}_0) \\ \quad \wedge\ \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2)\}. \end{array}$$

—

Finally, $\mathsf{i2} = \mathsf{i}_2$ is used to eliminate i_2 in the postcondition, and pure terms are rearranged in both the pre- and postconditions:

$$\{\exists\alpha_1, \alpha_2, \mathsf{i}_2.\ (\mathsf{lseg}\,\alpha_1\,(\mathsf{i1}, \mathsf{i}_2)\ *\ \mathsf{lseg}\,\alpha_2\,(\mathsf{i}_2, \mathsf{j}_0))$$
$$\quad \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2\}$$
$$\{\exists\alpha_1, \alpha_2, \mathsf{i}_2.\ (\mathsf{lseg}\,\alpha_1\,(\mathsf{i1}, \mathsf{i}_2) \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1)$$
$$\quad *\ (\mathsf{lseg}\,\alpha_2(\mathsf{i}_2, \mathsf{j}_0)$$
$$\quad \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2)\}$$
$$\mathsf{mergesort}(\mathsf{i1}, \mathsf{i2}; \mathsf{n1})\{\alpha_1, \mathsf{i}_2\}$$
$$\{\exists\alpha_1, \alpha_2, \mathsf{i}_2.\ (\exists\beta_1.\ \mathsf{lseg}\,\beta_1\,(\mathsf{i1}, -) \wedge \beta_1 {\sim} \alpha_1 \wedge \mathbf{ord}\,\beta_1 \wedge \mathsf{i2} = \mathsf{i}_2)$$
$$\quad *\ (\mathsf{lseg}\,\alpha_2(\mathsf{i}_2, \mathsf{j}_0)$$
$$\quad \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2)\}$$
$$\{\exists\alpha_1, \alpha_2, \beta_1.$$
$$\quad ((\mathsf{lseg}\,\beta_1\,(\mathsf{i1}, -)\ *\ (\mathsf{lseg}\,\alpha_2\,(\mathsf{i2}, \mathsf{j}_0)) \wedge \beta_1 \sim \alpha_1 \wedge \mathbf{ord}\,\beta_1$$
$$\quad \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1 \wedge \alpha = \alpha_1{\cdot}\alpha_2\}.$$

—

merge with $\mathbf{goto}$'s

$$
\begin{array}{l}
\mathsf{merge}(\mathsf{i};\mathsf{n1},\mathsf{n2},\mathsf{i1},\mathsf{i2})\{\beta_1,\beta_2\} = \\
\quad \mathbf{newvar}\ \mathsf{a1}\ \mathbf{in}\ \mathbf{newvar}\ \mathsf{a2}\ \mathbf{in}\ \mathbf{newvar}\ \mathsf{j}\ \mathbf{in} \\
\quad \big(\mathsf{a1} := [\mathsf{i1}]\ ;\ \mathsf{a2} := [\mathsf{i2}]\ ; \\
\quad \mathbf{if}\ \mathsf{a1} \leq \mathsf{a2}\ \mathbf{then}\ \mathsf{i} := \mathsf{i1}\ ;\ \mathbf{goto}\ \ell\mathsf{1}\ \mathbf{else}\ \mathsf{i} := \mathsf{i2}\ ;\ \mathbf{goto}\ \ell\mathsf{2}\ ; \\
\ell\mathsf{1}\colon \mathbf{if}\ \mathsf{n1} = 1\ \mathbf{then}\ [\mathsf{i1}+1] := \mathsf{i2}\ ;\ \mathbf{goto}\ \mathsf{out}\ \mathbf{else} \\
\quad \mathsf{n1} := \mathsf{n1} - 1\ ;\ \mathsf{j} := \mathsf{i1}\ ;\ \mathsf{i1} := [\mathsf{j}+1]\ ;\ \mathsf{a1} := [\mathsf{i1}]; \\
\quad \mathbf{if}\ \mathsf{a1} \leq \mathsf{a2}\ \mathbf{then}\ \mathbf{goto}\ \ell\mathsf{1}\ \mathbf{else}\ [\mathsf{j}+1] := \mathsf{i2}\ ;\ \mathbf{goto}\ \ell\mathsf{2}\ ; \\
\ell\mathsf{2}\colon \mathbf{if}\ \mathsf{n2} = 1\ \mathbf{then}\ [\mathsf{i2}+1] := \mathsf{i1}\ ;\ \mathbf{goto}\ \mathsf{out}\ \mathbf{else} \\
\quad \mathsf{n2} := \mathsf{n2} - 1\ ;\ \mathsf{j} := \mathsf{i2}\ ;\ \mathsf{i2} := [\mathsf{j}+1]\ ;\ \mathsf{a2} := [\mathsf{i2}]; \\
\quad \mathbf{if}\ \mathsf{a2} \leq \mathsf{a1}\ \mathbf{then}\ \mathbf{goto}\ \ell\mathsf{2}\ \mathbf{else}\ [\mathsf{j}+1] := \mathsf{i1}\ ;\ \mathbf{goto}\ \ell\mathsf{1}\ ; \\
\mathsf{out}\colon \big)
\end{array}
$$

—

A Proof for merge with goto's

$$\begin{array}{l}
\mathsf{merge}(\mathsf{i};\mathsf{n1},\mathsf{n2},\mathsf{i1},\mathsf{i2})\{\beta_1,\beta_2\} = \\
\quad \{(\mathsf{lseg}\,\beta_1\,(\mathsf{i1},-) \wedge \mathbf{ord}\,\beta_1 \wedge \#\beta_1 = \mathsf{n1} \wedge \mathsf{n1}\geq 1) \\
\quad\quad * (\mathsf{lseg}\,\beta_2\,(\mathsf{i2},-) \wedge \mathbf{ord}\,\beta_2 \wedge \#\beta_2 = \mathsf{n2} \wedge \mathsf{n2}\geq 1)\} \\
\quad \mathbf{newvar}\ \mathsf{a1}\ \mathbf{in}\ \mathbf{newvar}\ \mathsf{a2}\ \mathbf{in}\ \mathbf{newvar}\ \mathsf{j}\ \mathbf{in} \\
\quad\ \big(\mathsf{a1} := [\mathsf{i1}]\ ;\ \mathsf{a2} := [\mathsf{i2}]\ ; \\
\quad\ \mathbf{if}\ \mathsf{a1} \leq \mathsf{a2}\ \mathbf{then}\ \mathsf{i} := \mathsf{i1}\ ;\ \mathbf{goto}\ \ell 1\ \mathbf{else}\ \mathsf{i} := \mathsf{i2}\ ;\ \mathbf{goto}\ \ell 2\ ;
\end{array}$$

$$\begin{array}{l}
\ell 1\text{:}\ \{\exists \beta, \mathsf{a1}, \mathsf{j1}, \gamma_1, \mathsf{j2}, \gamma_2. \\
\quad\quad (\mathsf{lseg}\,\beta\,(\mathsf{i},\mathsf{i1})\ *\ \mathsf{i1} \mapsto \mathsf{a1},\mathsf{j1}\ *\ \mathsf{lseg}\,\gamma_1\,(\mathsf{j1},-) \\
\quad\quad\quad *\ \mathsf{i2} \mapsto \mathsf{a2},\mathsf{j2}\ *\ \mathsf{lseg}\,\gamma_2\,(\mathsf{j2},-)) \\
\quad\quad \wedge \#\gamma_1 = \mathsf{n1} - 1 \wedge \#\gamma_2 = \mathsf{n2} - 1 \\
\quad\quad \wedge \beta{\cdot}\mathsf{a1}{\cdot}\gamma_1{\cdot}\mathsf{a2}{\cdot}\gamma_2 {\sim} \beta_1{\cdot}\beta_2 \wedge \mathbf{ord}\,(\mathsf{a1}{\cdot}\gamma_1) \wedge \mathbf{ord}\,(\mathsf{a2}{\cdot}\gamma_2) \\
\quad\quad \wedge\, \mathbf{ord}\,\beta \wedge \{\beta\} \leq^* \{\mathsf{a1}{\cdot}\gamma_1\} \cup \{\mathsf{a2}{\cdot}\gamma_2\} \wedge \mathsf{a1} \leq \mathsf{a2}\} \\
\quad \mathbf{if}\ \mathsf{n1} = 1\ \mathbf{then}\ [\mathsf{i1} + 1] := \mathsf{i2}\ ;\ \mathbf{goto}\ \mathsf{out}\ \mathbf{else} \\
\quad \mathsf{n1} := \mathsf{n1} - 1\ ;\ \mathsf{j} := \mathsf{i1}\ ;\ \mathsf{i1} := [\mathsf{j} + 1]\ ;\ \mathsf{a1} := [\mathsf{i1}]; \\
\quad \{\exists \beta, \mathsf{a1}', \mathsf{j1}, \gamma_1', \mathsf{j2}, \gamma_2. \\
\quad\quad (\mathsf{lseg}\,\beta\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a1}',\mathsf{i1}\ *\ \mathsf{i1} \mapsto \mathsf{a1},\mathsf{j1}\ *\ \mathsf{lseg}\,\gamma_1'\,(\mathsf{j1},-) \\
\quad\quad\quad *\ \mathsf{i2} \mapsto \mathsf{a2},\mathsf{j2}\ *\ \mathsf{lseg}\,\gamma_2\,(\mathsf{j2},-)) \\
\quad\quad \wedge \#\gamma_1' = \mathsf{n1} - 1 \wedge \#\gamma_2 = \mathsf{n2} - 1 \\
\quad\quad \wedge \beta{\cdot}\mathsf{a1}'{\cdot}\mathsf{a1}{\cdot}\gamma_1'{\cdot}\mathsf{a2}{\cdot}\gamma_2 {\sim} \beta_1{\cdot}\beta_2 \wedge \mathbf{ord}\,(\mathsf{a1}'{\cdot}\mathsf{a1}{\cdot}\gamma_1') \wedge \mathbf{ord}\,(\mathsf{a2}{\cdot}\gamma_2) \\
\quad\quad \wedge\, \mathbf{ord}\,\beta \wedge \{\beta\} \leq^* \{\mathsf{a1}'{\cdot}\mathsf{a1}{\cdot}\gamma_1'\} \cup \{\mathsf{a2}{\cdot}\gamma_2\} \wedge \mathsf{a1}' \leq \mathsf{a2}\} \\
\quad \mathbf{if}\ \mathsf{a1} \leq \mathsf{a2}\ \mathbf{then}\ \mathbf{goto}\ \ell 1\ \mathbf{else}\ [\mathsf{j} + 1] := \mathsf{i2}\ ;\ \mathbf{goto}\ \ell 2\ ; \\
\quad\quad \vdots
\end{array}$$

—

$\vdots$

$\ell 2$: $\{\exists \beta, \mathsf{a2}, \mathsf{j2}, \gamma_2, \mathsf{j1}, \gamma_1.$

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

$*\ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1}\ *\ \mathsf{lseg}\ \gamma_1\ (\mathsf{j1}, -))$

$\wedge \#\gamma_2 = \mathsf{n2} - 1 \wedge \#\gamma_1 = \mathsf{n1} - 1$

$\wedge\ \beta \cdot \mathsf{a2} \cdot \gamma_2 \cdot \mathsf{a1} \cdot \gamma_1 \sim \beta_2 \cdot \beta_1 \wedge \mathbf{ord}\ (\mathsf{a2} \cdot \gamma_2) \wedge \mathbf{ord}\ (\mathsf{a1} \cdot \gamma_1)$

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

if n2 = 1 **then** [i2 + 1] := i1 ; **goto** out **else**

n2 := n2 − 1 ; j := i2 ; i2 := [j + 1] ; a2 := [i2];

$\{\exists \beta, \mathsf{a2}', \mathsf{j2}, \gamma_2', \mathsf{j1}, \gamma_1.$

$(\mathsf{lseg}\ \beta\ (\mathsf{i}, \mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a2}', \mathsf{i2}\ *\ \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2}\ *\ \mathsf{lseg}\ \gamma_2'\ (\mathsf{j2}, -)$

$*\ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1}\ *\ \mathsf{lseg}\ \gamma_1\ (\mathsf{j1}, -))$

$\wedge \#\gamma_2' = \mathsf{n2} - 1 \wedge \#\gamma_1 = \mathsf{n1} - 1$

$\wedge\ \beta \cdot \mathsf{a2}' \cdot \mathsf{a2} \cdot \gamma_2' \cdot \mathsf{a1} \cdot \gamma_1 \sim \beta_2 \cdot \beta_1 \wedge \mathbf{ord}\ (\mathsf{a2}' \cdot \mathsf{a2} \cdot \gamma_2') \wedge \mathbf{ord}\ (\mathsf{a1} \cdot \gamma_1)$

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

if a2 ≤ a1 **then goto** $\ell 2$ **else** [j + 1] := i1 ; **goto** $\ell 1$;

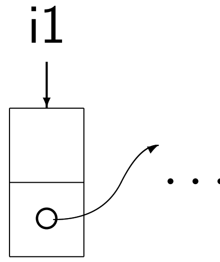
out:)

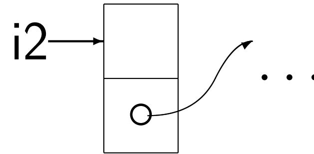
$\{\exists \beta.\ \mathsf{lseg}\ \beta\ (\mathsf{i}, -) \wedge \beta \sim \beta_1 \cdot \beta_2 \wedge \mathbf{ord}\ \beta\}.$

—

1.

β_1

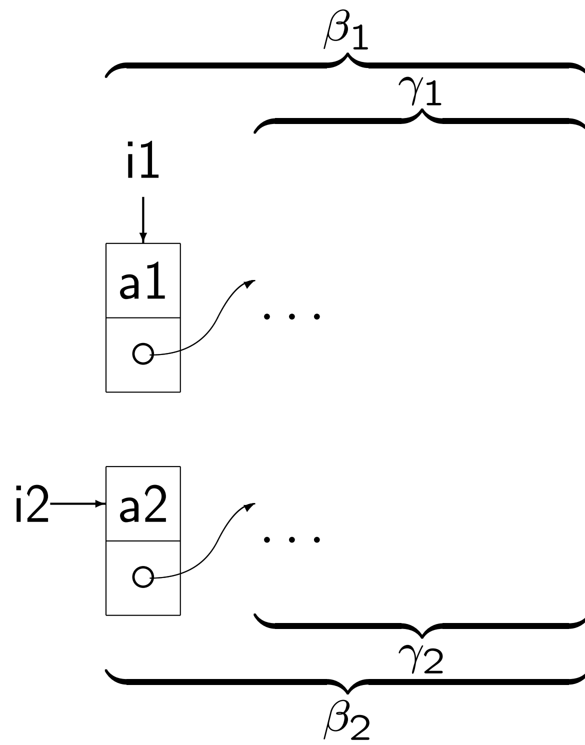




β_2

$$\{(\mathsf{lseg}\ \beta_1\,(\mathsf{i1}, -) \wedge \mathbf{ord}\ \beta_1 \wedge \#\beta_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1)$$
$$*\,(\mathsf{lseg}\ \beta_2\,(\mathsf{i2}, -) \wedge \mathbf{ord}\ \beta_2 \wedge \#\beta_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1)\}$$

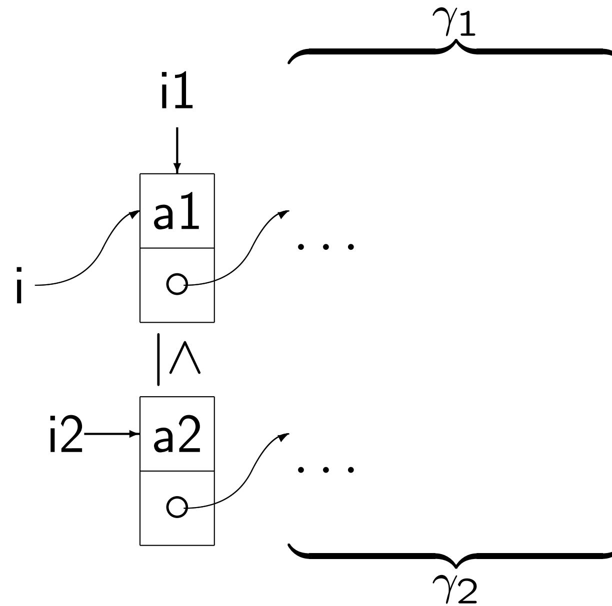
2.



$$
\begin{array}{l}
\textbf{newvar}\ \mathsf{a1}\ \textbf{in}\ \textbf{newvar}\ \mathsf{a2}\ \textbf{in}\ \textbf{newvar}\ \mathsf{j}\ \textbf{in} \\
\ \big(\mathsf{a1} := [\mathsf{i1}]\ ;\ \mathsf{a2} := [\mathsf{i2}]\ ; \\
\ \{\exists \mathsf{j1}, \gamma_1, \mathsf{j2}, \gamma_2. \\
\quad (\mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1}\ *\ \mathsf{lseg}\ \gamma_1\ (\mathsf{j1}, -) \\
\qquad *\ \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2}\ *\ \mathsf{lseg}\ \gamma_2\ (\mathsf{j2}, -)) \\
\quad\ \wedge \#\gamma_1 = \mathsf{n1} - 1 \wedge \#\gamma_2 = \mathsf{n2} - 1 \\
\quad\ \wedge\ \mathsf{a1}{\cdot}\gamma_1{\cdot}\mathsf{a2}{\cdot}\gamma_2 = \beta_1{\cdot}\beta_2 \wedge \mathbf{ord}\,(\mathsf{a1}{\cdot}\gamma_1) \wedge \mathbf{ord}\,(\mathsf{a2}{\cdot}\gamma_2)\}
\end{array}
$$

—

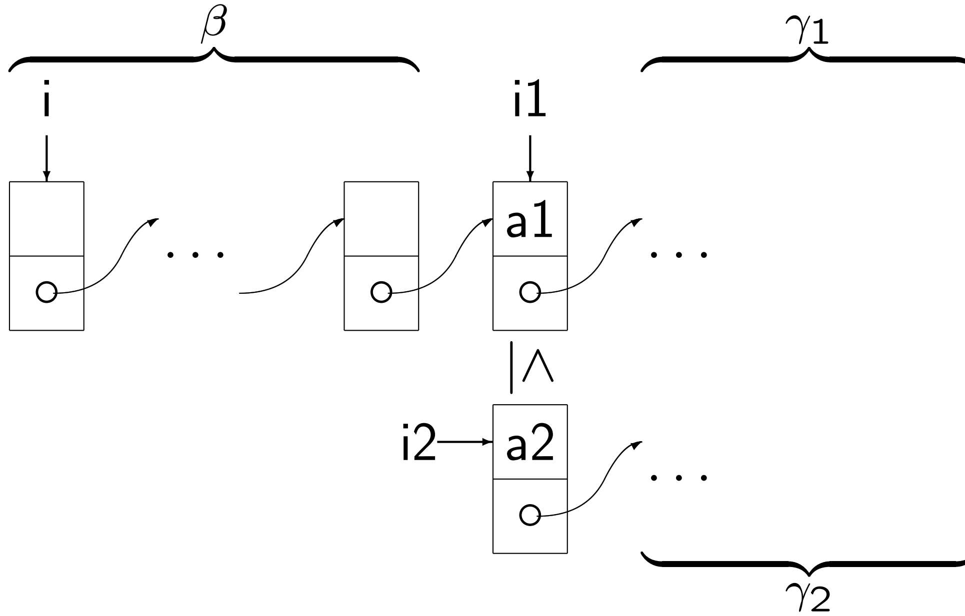
3.



$$
\begin{array}{l}
\textbf{newvar}\ \mathsf{a1}\ \textbf{in}\ \textbf{newvar}\ \mathsf{a2}\ \textbf{in}\ \textbf{newvar}\ \mathsf{j}\ \textbf{in} \\
\big(\mathsf{a1} := [\mathsf{i1}]\ ;\ \mathsf{a2} := [\mathsf{i2}]\ ; \\
\textbf{if}\ \mathsf{a1} \leq \mathsf{a2}\ \textbf{then}\ \mathsf{i} := \mathsf{i1}; \\
\{\exists \mathsf{a1}, \mathsf{j1}, \gamma_1, \mathsf{j2}, \gamma_2. \\
\quad ((\mathsf{i} = \mathsf{i1} \wedge \mathbf{emp})\ *\ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1}\ *\ \mathsf{lseg}\ \gamma_1\ (\mathsf{j1}, -) \\
\qquad *\ \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2}\ *\ \mathsf{lseg}\ \gamma_2\ (\mathsf{j2}, -)) \\
\quad \wedge \#\gamma_1 = \mathsf{n1} - 1 \wedge \#\gamma_2 = \mathsf{n2} - 1 \\
\quad \wedge \mathsf{a1}{\cdot}\gamma_1{\cdot}\mathsf{a2}{\cdot}\gamma_2 = \beta_1{\cdot}\beta_2 \wedge \mathbf{ord}\ (\mathsf{a1}{\cdot}\gamma_1) \wedge \mathbf{ord}\ (\mathsf{a2}{\cdot}\gamma_2) \\
\quad \wedge \mathsf{a1} \leq \mathsf{a2}\} \\
\textbf{goto}\ \ell 1
\end{array}
$$

—

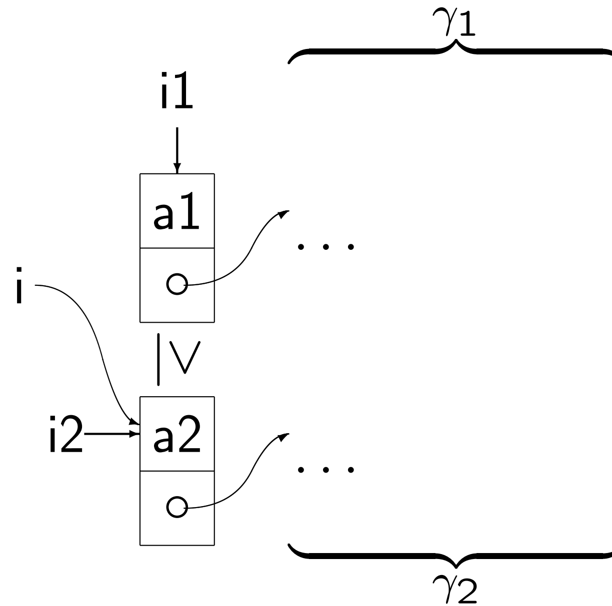
4.



$$
\begin{aligned}
&\ell 1\text{: } \{\exists \beta, \mathsf{a1}, \mathsf{j1}, \gamma_1, \mathsf{j2}, \gamma_2. \\
&\quad (\mathsf{lseg}\, \beta\, (\mathsf{i}, \mathsf{i1}) \ * \ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1} \ * \ \mathsf{lseg}\, \gamma_1\, (\mathsf{j1}, -) \\
&\qquad * \ \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2} \ * \ \mathsf{lseg}\, \gamma_2\, (\mathsf{j2}, -)) \\
&\quad \wedge \#\gamma_1 = \mathsf{n1} - 1 \wedge \#\gamma_2 = \mathsf{n2} - 1 \\
&\quad \wedge \beta{\cdot}\mathsf{a1}{\cdot}\gamma_1{\cdot}\mathsf{a2}{\cdot}\gamma_2 {\sim} \beta_1{\cdot}\beta_2 \wedge \mathbf{ord}\, (\mathsf{a1}{\cdot}\gamma_1) \wedge \mathbf{ord}\, (\mathsf{a2}{\cdot}\gamma_2) \\
&\quad \wedge \mathbf{ord}\, \beta \wedge \{\beta\} \leq^* \{\mathsf{a1}{\cdot}\gamma_1\} \cup \{\mathsf{a2}{\cdot}\gamma_2\} \wedge \mathsf{a1} \leq \mathsf{a2}\}
\end{aligned}
$$

—

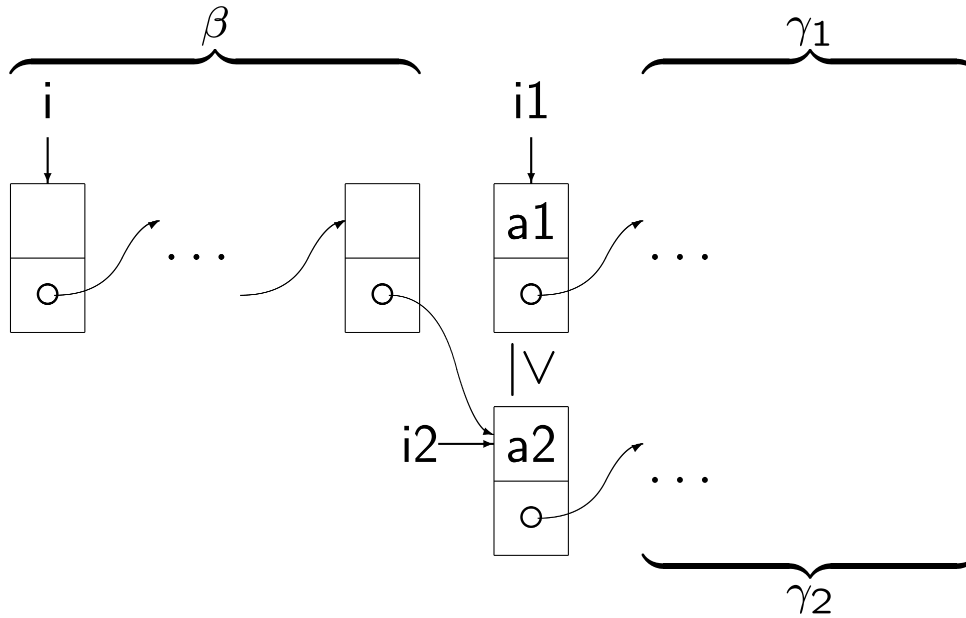
5.



$$
\begin{array}{l}
\textbf{newvar } \mathsf{a1} \textbf{ in newvar } \mathsf{a2} \textbf{ in newvar } \mathsf{j} \textbf{ in} \\
\big(\mathsf{a1} := [\mathsf{i1}]\ ;\ \mathsf{a2} := [\mathsf{i2}]\ ; \\
\textbf{if } \mathsf{a1} \leq \mathsf{a2} \textbf{ then } \mathsf{i} := \mathsf{i1}\ ;\ \textbf{goto } \ell 1 \textbf{ else } \mathsf{i} := \mathsf{i2}; \\
\{\exists \mathsf{a2}, \mathsf{j1}, \gamma_1, \mathsf{j2}, \gamma_2. \\
\quad ((\mathsf{i} = \mathsf{i2} \wedge \mathbf{emp})\ *\ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1}\ *\ \mathsf{lseg}\ \gamma_1\ (\mathsf{j1}, -) \\
\qquad *\ \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2}\ *\ \mathsf{lseg}\ \gamma_2\ (\mathsf{j2}, -)) \\
\quad \wedge \#\gamma_1 = \mathsf{n1} - 1 \wedge \#\gamma_2 = \mathsf{n2} - 1 \\
\quad \wedge \mathsf{a1}{\cdot}\gamma_1{\cdot}\mathsf{a2}{\cdot}\gamma_2 = \beta_1{\cdot}\beta_2 \wedge \mathbf{ord}\ (\mathsf{a1}{\cdot}\gamma_1) \wedge \mathbf{ord}\ (\mathsf{a2}{\cdot}\gamma_2) \\
\quad \wedge \mathsf{a2} \leq \mathsf{a1}\} \\
\textbf{goto } \ell 2\ ;
\end{array}
$$

—

6.



ℓ2: $\{\exists\beta, \mathsf{a2}, \mathsf{j2}, \gamma_2, \mathsf{j1}, \gamma_1.$

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

$*\ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1}\ *\ \mathsf{lseg}\ \gamma_1\ (\mathsf{j1}, -))$

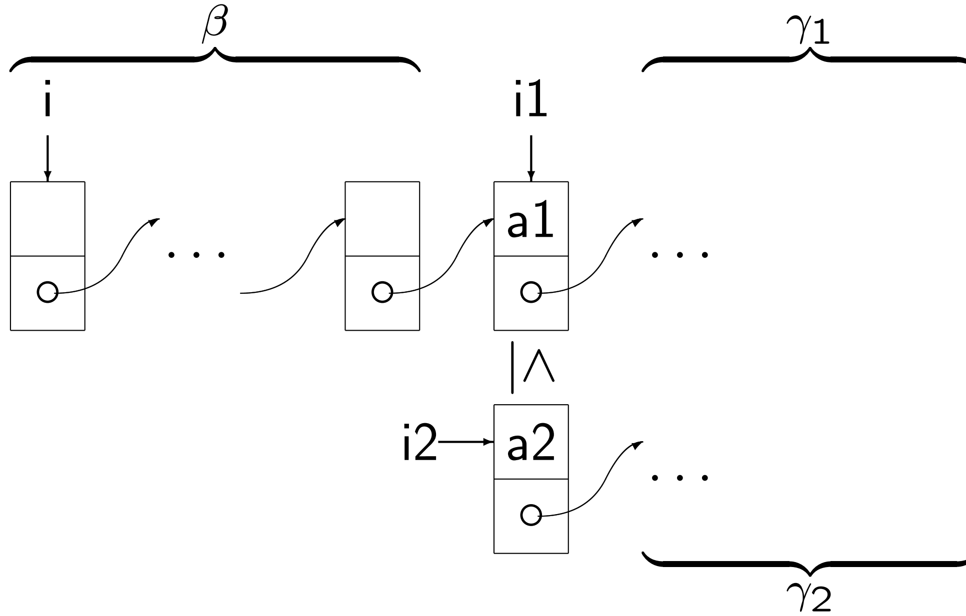
$\wedge\ \#\gamma_2 = \mathsf{n2} - 1 \wedge \#\gamma_1 = \mathsf{n1} - 1$

$\wedge\ \beta{\cdot}\mathsf{a2}{\cdot}\gamma_2{\cdot}\mathsf{a1}{\cdot}\gamma_1 {\sim} \beta_2{\cdot}\beta_1 \wedge \mathbf{ord}\ (\mathsf{a2}{\cdot}\gamma_2) \wedge \mathbf{ord}\ (\mathsf{a1}{\cdot}\gamma_1)$

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

—

7.



$\ell 1$: $\{\exists \beta, \mathsf{a1}, \mathsf{j1}, \gamma_1, \mathsf{j2}, \gamma_2.$

$(\mathsf{lseg}\ \beta\ (\mathsf{i}, \mathsf{i1})\ *\ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1}\ *\ \mathsf{lseg}\ \gamma_1\ (\mathsf{j1}, -)$

$*\ \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2}\ *\ \mathsf{lseg}\ \gamma_2\ (\mathsf{j2}, -))$

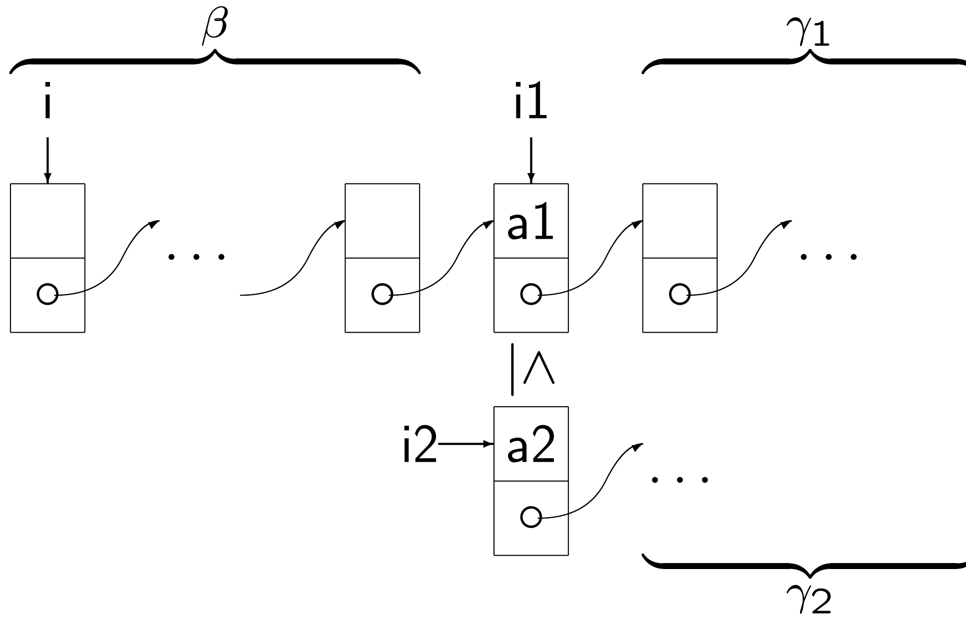
$\wedge\ \#\gamma_1 = \mathsf{n1} - 1 \wedge \#\gamma_2 = \mathsf{n2} - 1$

$\wedge\ \beta{\cdot}\mathsf{a1}{\cdot}\gamma_1{\cdot}\mathsf{a2}{\cdot}\gamma_2 \sim \beta_1{\cdot}\beta_2 \wedge \mathbf{ord}\ (\mathsf{a1}{\cdot}\gamma_1) \wedge \mathbf{ord}\ (\mathsf{a2}{\cdot}\gamma_2)$

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

—

8.

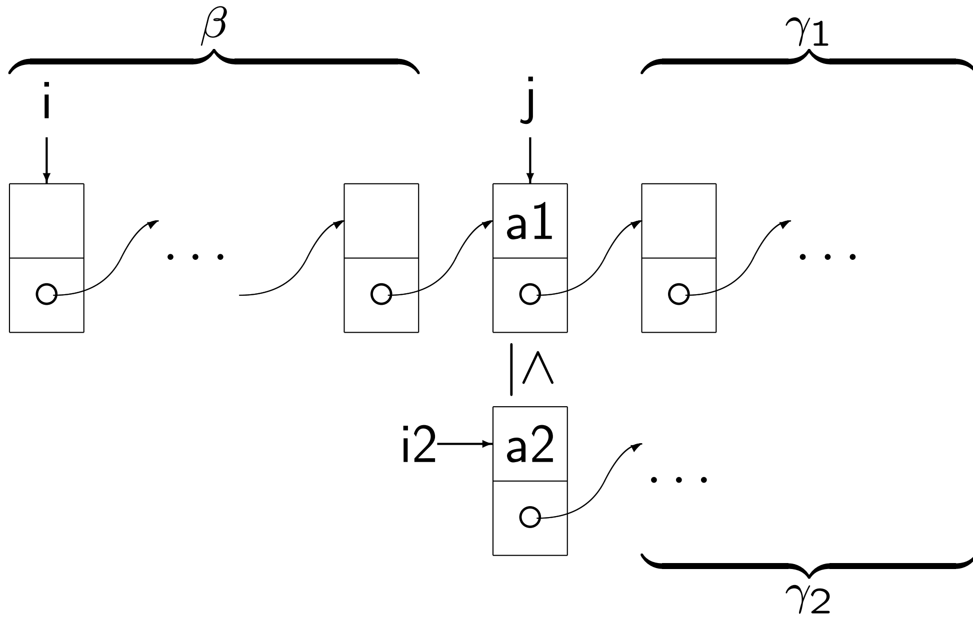


if $\mathsf{n1} = 1$ **then** $[\mathsf{i1} + 1] := \mathsf{i2}$; **goto** out **else**

$\mathsf{n1} := \mathsf{n1} - 1$;

—

9.

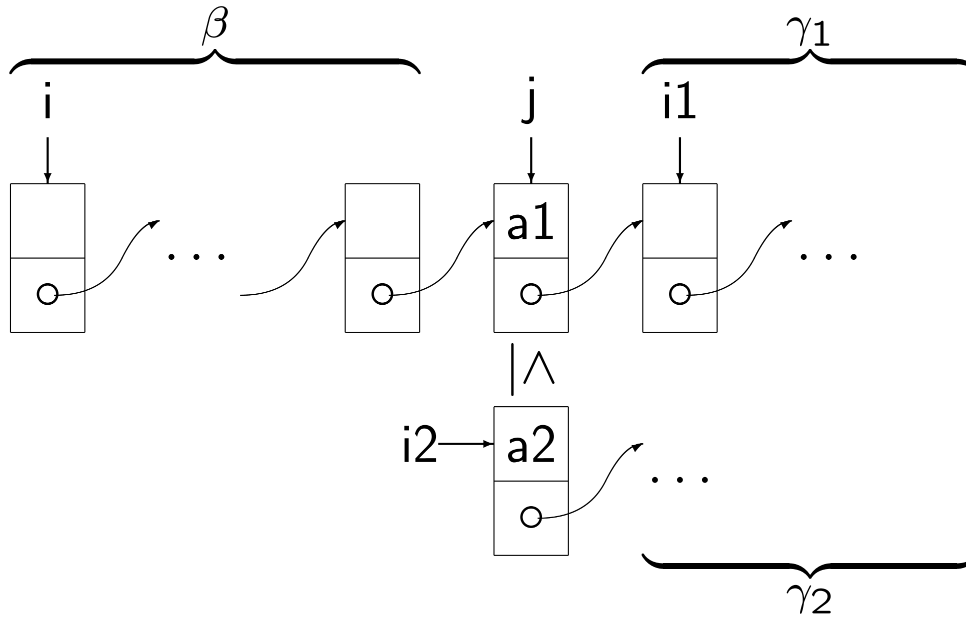


if n1 = 1 **then** [i1 + 1] := i2 ; **goto** out **else**

n1 := n1 − 1 ; j := i1;

—

10.

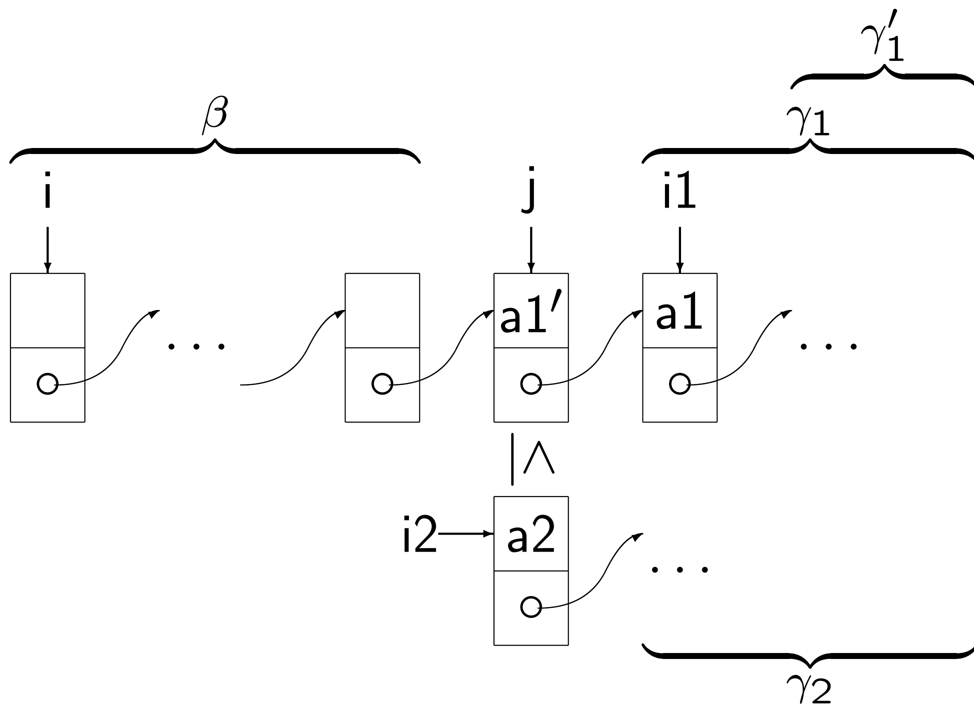


if n1 = 1 **then** [i1 + 1] := i2 ; **goto** out **else**

n1 := n1 − 1 ; j := i1 ; i1 := [j + 1];

—

11.



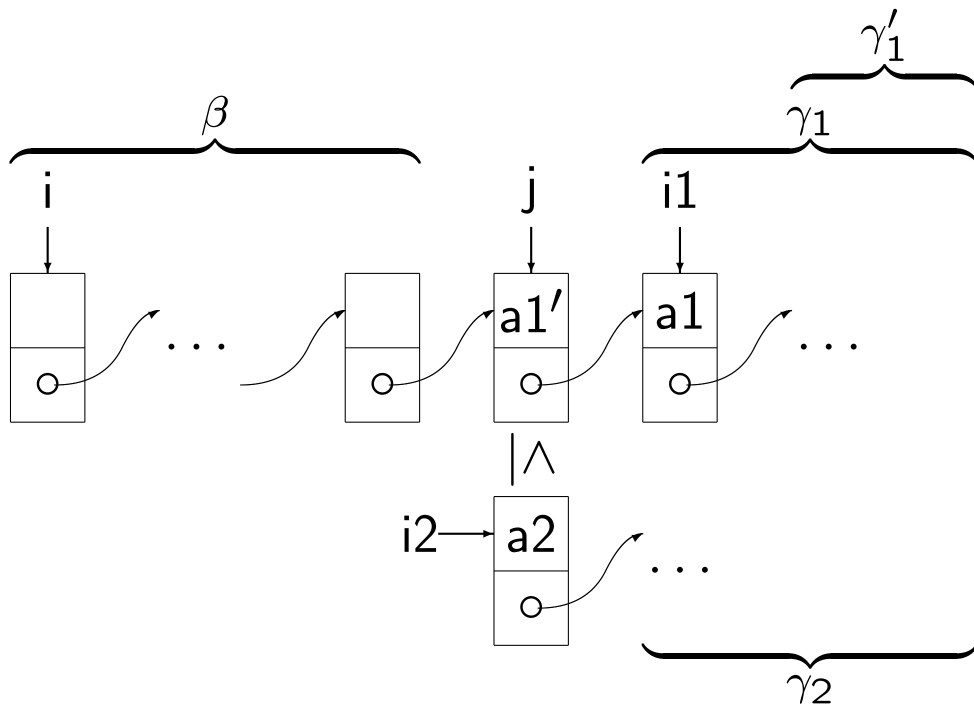
$\mathbf{if}\ \mathsf{n1} = 1\ \mathbf{then}\ [\mathsf{i1}+1] := \mathsf{i2}\ ;\ \mathbf{goto}\ \mathsf{out}\ \mathbf{else}$

$\mathsf{n1} := \mathsf{n1} - 1\ ;\ \mathsf{j} := \mathsf{i1}\ ;\ \mathsf{i1} := [\mathsf{j}+1]\ ;\ \mathsf{a1} := [\mathsf{i1}];$

$$\begin{aligned}
&\{\exists \beta, \mathsf{a1}', \mathsf{j1}, \gamma_1', \mathsf{j2}, \gamma_2. \\
&\quad (\mathsf{lseg}\ \beta\,(\mathsf{i},\mathsf{j}) \ *\ \mathsf{j} \mapsto \mathsf{a1}', \mathsf{i1} \ *\ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1} \ *\ \mathsf{lseg}\ \gamma_1'\,(\mathsf{j1}, -) \\
&\qquad *\ \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2} \ *\ \mathsf{lseg}\ \gamma_2\,(\mathsf{j2}, -)) \\
&\quad \wedge \#\gamma_1' = \mathsf{n1} - 1 \wedge \#\gamma_2 = \mathsf{n2} - 1 \\
&\quad \wedge \beta{\cdot}\mathsf{a1}'{\cdot}\mathsf{a1}{\cdot}\gamma_1'{\cdot}\mathsf{a2}{\cdot}\gamma_2 {\sim} \beta_1{\cdot}\beta_2 \\
&\quad \left.\begin{aligned}
&\wedge \mathbf{ord}\,(\mathsf{a1}'{\cdot}\mathsf{a1}{\cdot}\gamma_1') \wedge \mathbf{ord}\,(\mathsf{a2}{\cdot}\gamma_2) \wedge \mathbf{ord}\,\beta \\
&\wedge \{\beta\} \leq^* \{\mathsf{a1}'{\cdot}\mathsf{a1}{\cdot}\gamma_1'\} \cup \{\mathsf{a2}{\cdot}\gamma_2\} \wedge \mathsf{a1}' \leq \mathsf{a2}\}
\end{aligned}\right\}(A)
\end{aligned}$$

—

12.



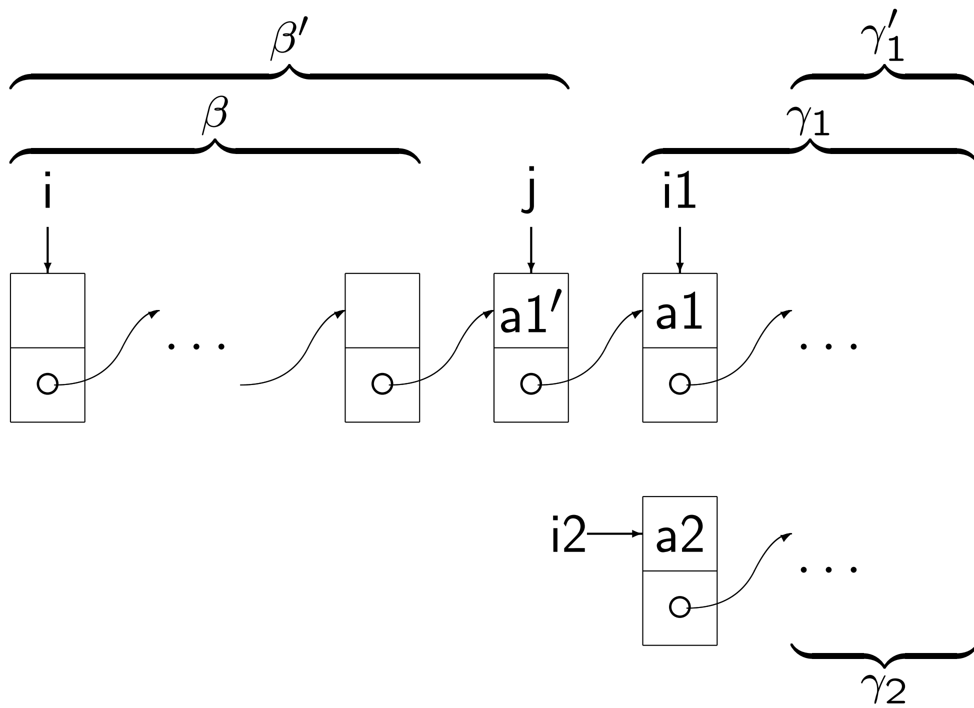
$\mathbf{if}\ \mathsf{n1} = 1\ \mathbf{then}\ [\mathsf{i1} + 1] := \mathsf{i2}\ ;\ \mathbf{goto}\ \mathsf{out}\ \mathbf{else}$

$\mathsf{n1} := \mathsf{n1} - 1\ ;\ \mathsf{j} := \mathsf{i1}\ ;\ \mathsf{i1} := [\mathsf{j} + 1]\ ;\ \mathsf{a1} := [\mathsf{i1}];$

$$
\begin{aligned}
&\{\exists \beta, \mathsf{a1}', \mathsf{j1}, \gamma_1', \mathsf{j2}, \gamma_2.\\
&\quad (\mathsf{lseg}\ \beta\ (\mathsf{i}, \mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a1}', \mathsf{i1}\ *\ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1}\ *\ \mathsf{lseg}\ \gamma_1'\ (\mathsf{j1}, -)\\
&\qquad *\ \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2}\ *\ \mathsf{lseg}\ \gamma_2\ (\mathsf{j2}, -))\\
&\quad \wedge \#\gamma_1' = \mathsf{n1} - 1 \wedge \#\gamma_2 = \mathsf{n2} - 1\\
&\quad \wedge \beta \cdot \mathsf{a1}' \cdot \mathsf{a1} \cdot \gamma_1' \cdot \mathsf{a2} \cdot \gamma_2 \sim \beta_1 \cdot \beta_2\\
&\quad \left.\begin{aligned}
&\wedge \mathbf{ord}\,(\mathsf{a1} \cdot \gamma_1') \wedge \mathbf{ord}\,(\mathsf{a2} \cdot \gamma_2) \wedge \mathbf{ord}\,(\beta \cdot \mathsf{a1}')\\
&\wedge \{\beta \cdot \mathsf{a1}'\} \leq^* \{\mathsf{a1} \cdot \gamma_1'\} \cup \{\mathsf{a2} \cdot \gamma_2\}\}
\end{aligned}\right\}\ (B)
\end{aligned}
$$

—

13.



$\mathbf{if}\ \mathsf{n1} = 1\ \mathbf{then}\ [\mathsf{i1} + 1] := \mathsf{i2}\ ;\ \mathbf{goto}\ \mathsf{out}\ \mathbf{else}$

$\mathsf{n1} := \mathsf{n1} - 1\ ;\ \mathsf{j} := \mathsf{i1}\ ;\ \mathsf{i1} := [\mathsf{j} + 1]\ ;\ \mathsf{a1} := [\mathsf{i1}];$

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

$(\mathsf{lseg}\ \beta\ (\mathsf{i}, \mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a1}', \mathsf{i1}\ *\ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1}\ *\ \mathsf{lseg}\ \gamma_1'\ (\mathsf{j1}, -)$

$*\ \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2}\ *\ \mathsf{lseg}\ \gamma_2\ (\mathsf{j2}, -))$

$\wedge \#\gamma_1' = \mathsf{n1} - 1 \wedge \#\gamma_2 = \mathsf{n2} - 1$

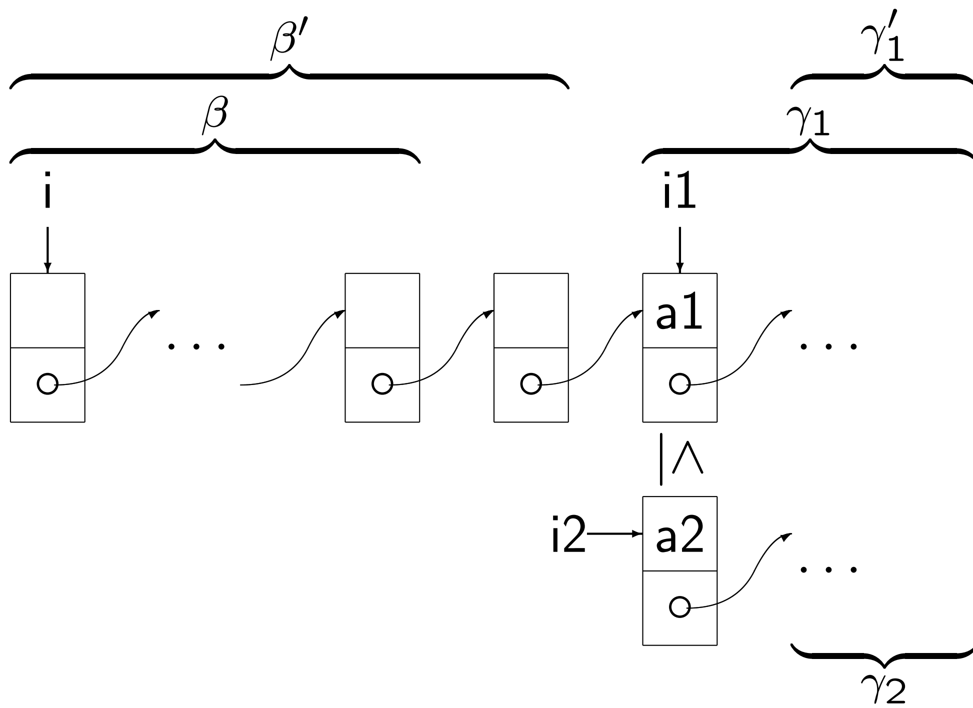
$\wedge \beta' = \beta{\cdot}\mathsf{a1}' \wedge \beta'{\cdot}\mathsf{a1}{\cdot}\gamma_1'{\cdot}\mathsf{a2}{\cdot}\gamma_2 {\sim} \beta_1{\cdot}\beta_2$

$\wedge\ \mathbf{ord}\ (\mathsf{a1}{\cdot}\gamma_1') \wedge \mathbf{ord}\ (\mathsf{a2}{\cdot}\gamma_2) \wedge \mathbf{ord}\ \beta'$

$\wedge \{\beta'\} \leq^* \{\mathsf{a1}{\cdot}\gamma_1'\} \cup \{\mathsf{a2}{\cdot}\gamma_2\}\}$

—

14.

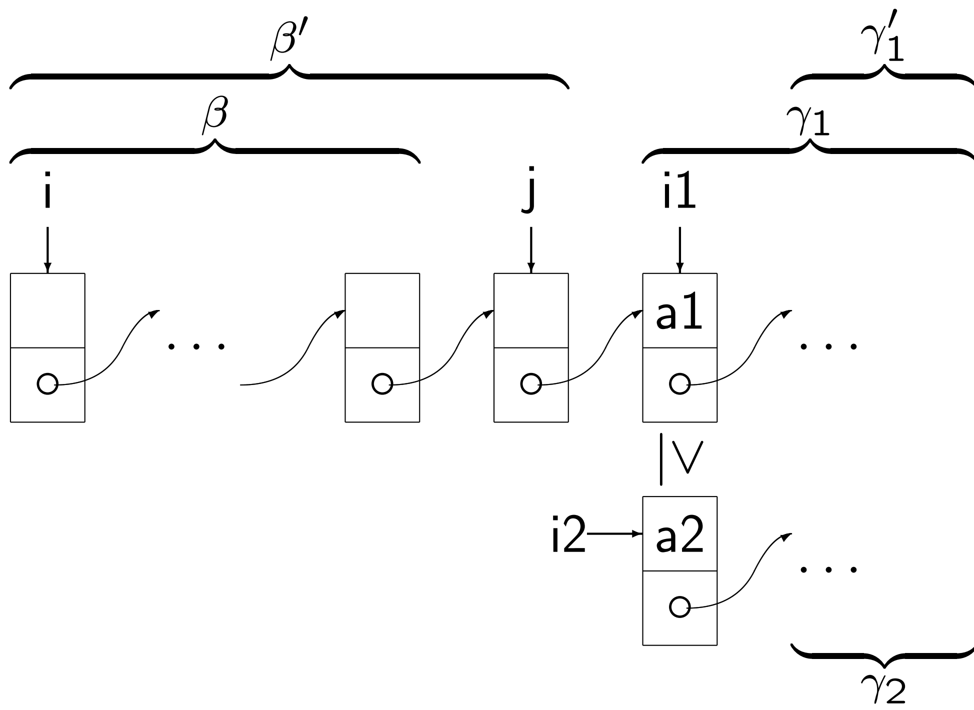


$$\textbf{if}\ \mathsf{a1} \leq \mathsf{a2}\ \textbf{then goto}\ \ell 1$$

$$\begin{aligned}
\ell 1\text{: } \{&\exists \beta', \mathsf{a1}, \mathsf{j1}, \gamma_1', \mathsf{j2}, \gamma_2. \\
&(\mathsf{lseg}\ \beta'\,(\mathsf{i}, \mathsf{i1})\ *\ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1}\ *\ \mathsf{lseg}\ \gamma_1'\,(\mathsf{j1}, -) \\
&\quad *\ \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2}\ *\ \mathsf{lseg}\ \gamma_2\,(\mathsf{j2}, -)) \\
&\wedge \#\gamma_1' = \mathsf{n1} - 1 \wedge \#\gamma_2 = \mathsf{n2} - 1 \\
&\wedge \beta' {\cdot} \mathsf{a1} {\cdot} \gamma_1' {\cdot} \mathsf{a2} {\cdot} \gamma_2 {\sim} \beta_1 {\cdot} \beta_2 \\
&\wedge \mathbf{ord}\,(\mathsf{a1} {\cdot} \gamma_1') \wedge \mathbf{ord}\,(\mathsf{a2} {\cdot} \gamma_2) \wedge \mathbf{ord}\ \beta' \\
&\wedge \{\beta'\} \leq^* \{\mathsf{a1} {\cdot} \gamma_1'\} \cup \{\mathsf{a2} {\cdot} \gamma_2\} \wedge \mathsf{a1} \leq \mathsf{a2}\}
\end{aligned}$$

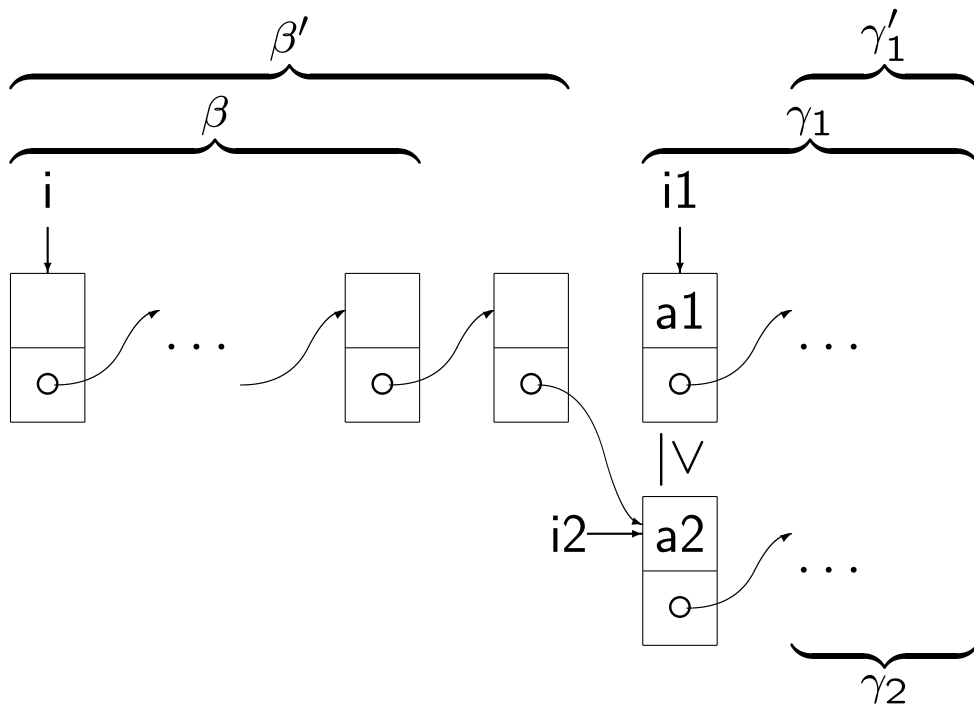
—

15.



if $\mathsf{a1} \leq \mathsf{a2}$ **then goto** $\ell 1$ **else**

16.



$$\textbf{if } \mathsf{a1} \leq \mathsf{a2} \textbf{ then goto } \ell 1 \textbf{ else } [\mathsf{j}+1] := \mathsf{i2} \, ; \textbf{goto } \ell 2 \, ;$$

$$\begin{aligned}
\ell 2\text{: } \{ & \exists \beta', \mathsf{a2}, \mathsf{j2}, \gamma_2', \mathsf{j1}, \gamma_1 . \\
& (\mathsf{lseg}\ \beta'\ (\mathsf{i}, \mathsf{i2}) \ * \ \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2} \ * \ \mathsf{lseg}\ \gamma_2'\ (\mathsf{j2}, -) \\
& \quad * \ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1} \ * \ \mathsf{lseg}\ \gamma_1\ (\mathsf{j1}, -)) \\
& \wedge \# \gamma_2' = \mathsf{n2} - 1 \wedge \# \gamma_1 = \mathsf{n1} - 1 \\
& \wedge \beta' {\cdot} \mathsf{a2} {\cdot} \gamma_2' {\cdot} \mathsf{a1} {\cdot} \gamma_1 {\sim} \beta_2 {\cdot} \beta_1 \\
& \wedge \mathbf{ord}\ (\mathsf{a2} {\cdot} \gamma_2') \wedge \mathbf{ord}\ (\mathsf{a1} {\cdot} \gamma_1) \wedge \mathbf{ord}\ \beta' \\
& \wedge \{\beta'\} \leq^* \{\mathsf{a2} {\cdot} \gamma_2'\} \cup \{\mathsf{a1} {\cdot} \gamma_1\} \wedge \mathsf{a2} \leq \mathsf{a1} \}
\end{aligned}$$

—

The Ordering Argument

$$\left.\begin{array}{l}\mathbf{ord}\,(\mathsf{a1}'\cdot\mathsf{a1}\cdot\gamma_1') \wedge \mathbf{ord}\,(\mathsf{a2}\cdot\gamma_2) \wedge \mathbf{ord}\,\beta \\ \quad \wedge \{\beta\} \leq^* \{\mathsf{a1}'\cdot\mathsf{a1}\cdot\gamma_1'\} \cup \{\mathsf{a2}\cdot\gamma_2\}\} \wedge \mathsf{a1}' \leq \mathsf{a2}\end{array}\right\}(A)$$

$$\Rightarrow$$

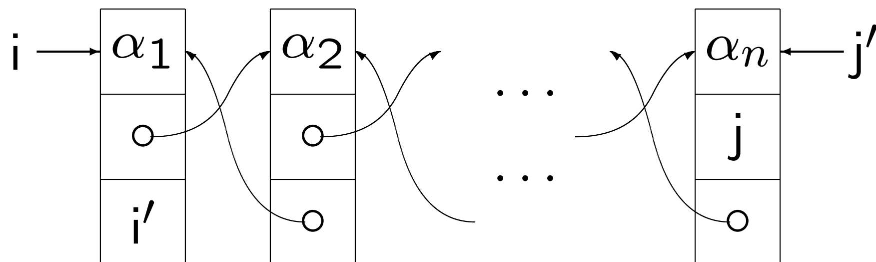
$$\left.\begin{array}{l}\mathbf{ord}\,(\mathsf{a1}\cdot\gamma_1') \wedge \mathbf{ord}\,(\mathsf{a2}\cdot\gamma_2) \wedge \mathbf{ord}\,(\beta\cdot\mathsf{a1}') \\ \quad \wedge \{\beta\cdot\mathsf{a1}'\} \leq^* \{\mathsf{a1}\cdot\gamma_1'\} \cup \{\mathsf{a2}\cdot\gamma_2\}\}\end{array}\right\}(B)$$

1.	$\mathbf{ord}\,(\mathsf{a1}'\cdot\mathsf{a1}\cdot\gamma_1')$	(assumption)
∗2.	$\mathbf{ord}\,(\mathsf{a1}\cdot\gamma_1')$	(13),1
3.	$\mathsf{a1}' \leq^* \{\mathsf{a1}\cdot\gamma_1'\}$	(13),1
4.	$\mathsf{a1}' \leq \mathsf{a2}$	(assumption)
∗5.	$\mathbf{ord}\,(\mathsf{a2}\cdot\gamma_2)$	(assumption)
6.	$\mathsf{a1}' \leq^* \{\mathsf{a2}\cdot\gamma_2\}$	(14),4,5
7.	$\mathsf{a1}' \leq^* \{\mathsf{a1}\cdot\gamma_1'\} \cup \{\mathsf{a2}\cdot\gamma_2\}$	(11),3,6
8.	$\{\beta\} \leq^* \{\mathsf{a1}'\cdot\mathsf{a1}\cdot\gamma_1'\} \cup \{\mathsf{a2}\cdot\gamma_2\}$	(assumption)
9.	$\{\beta\} \leq^* \{\mathsf{a1}\cdot\gamma_1'\} \cup \{\mathsf{a2}\cdot\gamma_2\}$	(3),(6),8
∗10.	$\{\beta\cdot\mathsf{a1}'\} \leq^* \{\mathsf{a1}\cdot\gamma_1'\} \cup \{\mathsf{a2}\cdot\gamma_2\}$	(10),(3),7,9
11.	$\mathbf{ord}\,\beta$	(assumption)
12.	$\mathbf{ord}\,\mathsf{a1}'$	(12)
13.	$\{\beta\} \leq^* \mathsf{a1}'$	(3),(6),8
∗14.	$\mathbf{ord}\,(\beta\cdot\mathsf{a1}')$	(13),11,12,13

—

Doubly-Linked List Segments

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

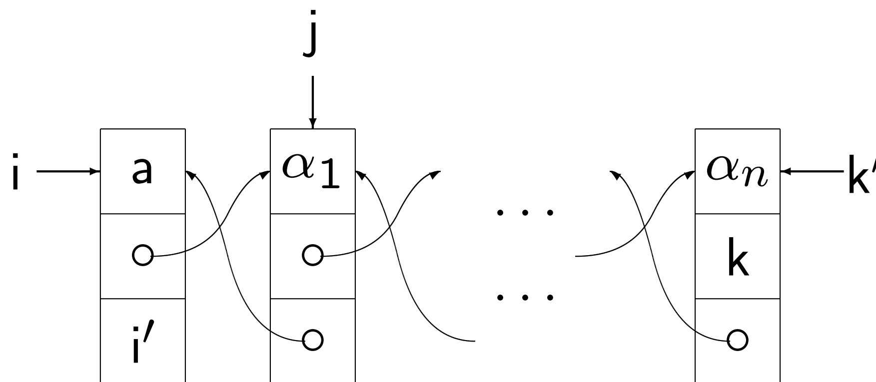


is defined by

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

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

The second of these equations is illustrated by:



—

Properties

$$\begin{aligned}\mathsf{dlseg}\ \mathsf{a}\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') &\Leftrightarrow \mathsf{i} \mapsto \mathsf{a}, \mathsf{j}, \mathsf{i}' \wedge \mathsf{i} = \mathsf{j}' \\ \mathsf{dlseg}\ \alpha{\cdot}\beta\ (\mathsf{i}, \mathsf{i}', \mathsf{k}, \mathsf{k}') &\Leftrightarrow \exists \mathsf{j}, \mathsf{j}'.\ \mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') * \mathsf{dlseg}\ \beta\ (\mathsf{j}, \mathsf{j}', \mathsf{k}, \mathsf{k}') \\ \mathsf{dlseg}\ \alpha{\cdot}\mathsf{b}\ (\mathsf{i}, \mathsf{i}', \mathsf{k}, \mathsf{k}') &\Leftrightarrow \exists \mathsf{j}'.\ \mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{k}', \mathsf{j}') * \mathsf{k}' \mapsto \mathsf{b}, \mathsf{k}, \mathsf{j}'.\end{aligned}$$

One can also define a doubly-linked list by

$$\mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathsf{j}') = \mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathbf{nil}, \mathsf{j}').$$

—

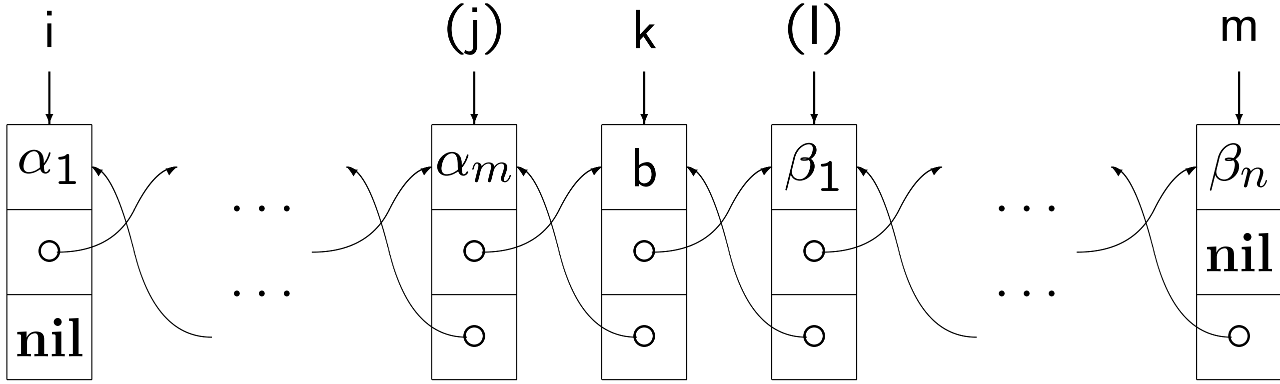
Emptyness Conditions

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

(One can also define nontouching segments.)

—

Deleting an Element from a Doubly-Linked List



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

$\mathsf{l} := [\mathsf{k}+1]\ ;\ \mathsf{j} := [\mathsf{k}+2]\ ;$

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

$\mathbf{dispose}\ \mathsf{k}\ ;\ \mathbf{dispose}\ \mathsf{k}+1\ ;\ \mathbf{dispose}\ \mathsf{k}+2\ ;$

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

$\mathbf{if}\ \mathsf{j} = \mathbf{nil}\ \mathbf{then}$

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

$\quad\mathsf{i} := \mathsf{l}$

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

$\mathbf{else}$

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

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

$\quad[\mathsf{j}+1] := \mathsf{l}\ ;$

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

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

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

$\quad\vdots$

$\vdots$

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

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

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

$\quad\mathsf{m} := \mathsf{j}$

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

$\mathbf{else}$

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

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

$\quad[\mathsf{l} + 2] := \mathsf{j}$

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

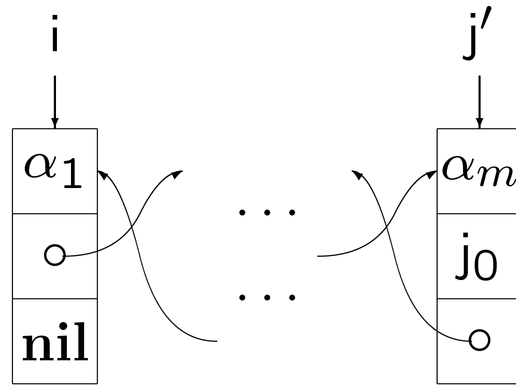
$\qquad *\ \mathsf{dlseg}\ \beta'\ (\mathsf{n}, \mathsf{l}, \mathbf{nil}, \mathsf{m})) \wedge \beta = \mathsf{a}{\cdot}\beta'\}$

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

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

—

A Lookup-Pointer Procedure for Doubly-Linked Lists



$\mathsf{lookuprpt}(\mathsf{j};\mathsf{i},\mathsf{j}')\{\alpha,\mathsf{j}_0\} =$

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

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

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

$\mathsf{j} := \mathsf{i}$

$\mathbf{else}$

$\{\exists\alpha',\mathsf{b},\mathsf{k}'.\ \ \alpha = \alpha'{\cdot}\mathsf{b} \wedge (\mathsf{dlseg}\ \alpha'\ (\mathsf{i},\mathbf{nil},\mathsf{j}',\mathsf{k}')\ *\ \mathsf{j}' \mapsto \mathsf{b},\mathsf{j}_0,\mathsf{k}')\}$

$\mathsf{j} := [\mathsf{j}' + 1]$

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

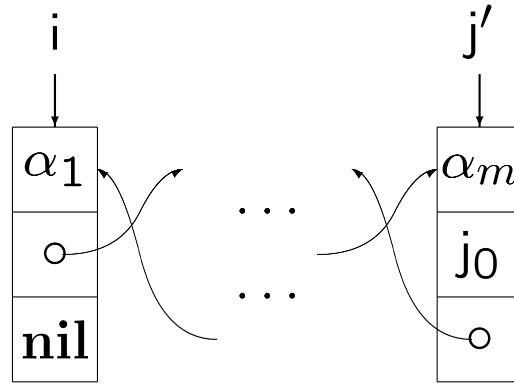
$(\mathsf{dlseg}\ \alpha'\ (\mathsf{i},\mathbf{nil},\mathsf{j}',\mathsf{k}')\ *\ \mathsf{j}' \mapsto \mathsf{b},\mathsf{j}_0,\mathsf{k}') \wedge \mathsf{j} = \mathsf{j}_0\}$

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

The parameter list here shows that the procedure call will only modify the variable j, and will not even evaluate the ghost parameters α or j_0. This information is an essential requirement for the procedure; otherwise the specification could be met by the assignment $\mathsf{j} := \mathsf{j}_0$.

—

A Set-Pointer Procedure



$\mathsf{setrpt}(\mathsf{i};\mathsf{j},\mathsf{j}')\{\alpha,\mathsf{j}_0\} =$

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

$\mathbf{if}\;\mathsf{j}' = \mathbf{nil}\;\mathbf{then}$

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

$\mathsf{i} := \mathsf{j}$

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

$\mathbf{else}$

$\{\exists\alpha', \mathsf{b}, \mathsf{k}'.\;\; \alpha = \alpha'{\cdot}\mathsf{b} \wedge (\mathsf{dlseg}\;\alpha'\;(\mathsf{i},\mathbf{nil},\mathsf{j}',\mathsf{k}') \;*\; \mathsf{j}' \mapsto \mathsf{b},\mathsf{j}_0,\mathsf{k}')\}$

$[\mathsf{j}' + 1] := \mathsf{j}$

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

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

—

An Oddity

From

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

we can infer, by weakening the consequent,

$$\{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j_0}, \mathsf{j'})\}$$
$$\mathsf{lookuprpt}(\mathsf{j}; \mathsf{i}, \mathsf{j'})\{\alpha, \mathsf{j_0}\}$$
$$\{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}, \mathsf{j'})\},$$

which is very similar to

$$\{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j_0}, \mathsf{j'})\}$$
$$\mathsf{setrpt}(\mathsf{i}; \mathsf{j}, \mathsf{j'})\{\alpha, \mathsf{j_0}\}$$
$$\{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}, \mathsf{j'})\}.$$

But the modified variables are different.

—

Upon Reflection

We can define a symmetric pair of procedures that act upon right-end segments:

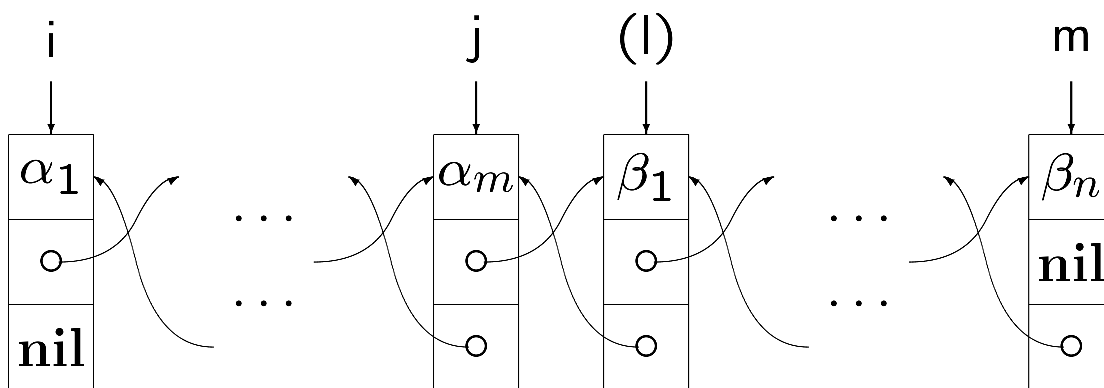
$$\begin{aligned}
&\mathsf{lookuplpt}(\mathsf{i}'; \mathsf{i}, \mathsf{j}')\{\alpha, \mathsf{i}'_0\} = \\
&\quad \{\mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathsf{i}'_0, \mathbf{nil}, \mathsf{j}')\} \\
&\quad \mathbf{if}\ \mathsf{i} = \mathbf{nil}\ \mathbf{then}\ \mathsf{i}' := \mathsf{j}'\ \mathbf{else}\ \mathsf{i}' := [\mathsf{i} + 2] \\
&\quad \{\mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathsf{i}'_0, \mathbf{nil}, \mathsf{j}') \wedge \mathsf{i}' = \mathsf{i}'_0\}
\end{aligned}$$

and

$$\begin{aligned}
&\mathsf{setlpt}(\mathsf{j}'; \mathsf{i}, \mathsf{i}')\{\alpha, \mathsf{i}'_0\} = \\
&\quad \{\mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathsf{i}'_0, \mathbf{nil}, \mathsf{j}')\} \\
&\quad \mathbf{if}\ \mathsf{i} = \mathbf{nil}\ \mathbf{then}\ \mathsf{j}' := \mathsf{i}'\ \mathbf{else}\ [\mathsf{i} + 2] := \mathsf{i}' \\
&\quad \{\mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathsf{i}', \mathbf{nil}, \mathsf{j}')\}.
\end{aligned}$$

—

Inserting an Element into a Doubly-Linked List



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

$\{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}_0, \mathsf{j}')\}\ \mathsf{setrpt}(\mathsf{i}; \mathsf{j}, \mathsf{j}')\{\alpha, \mathsf{j}_0\}\ \{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}, \mathsf{j}')\},$

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

$\{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathsf{i}'_0, \mathbf{nil}, \mathsf{j}')\ \ \mathsf{setlpt}(\mathsf{j}'; \mathsf{i}, \mathsf{i}')\{\alpha, \mathsf{i}'_0\}\ \ \{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathbf{nil}, \mathsf{j}')\} \vdash$

—

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

$$\left.\begin{array}{l}
\{\mathrm{dlseg}\,\alpha\,(i,\mathbf{nil},j_0,j) \ * \ \mathrm{dlseg}\,\beta\,(j_0,j,\mathbf{nil},m)\} \\
\quad \left.\begin{array}{l}
\{\mathrm{dlseg}\,\alpha\,(i,\mathbf{nil},j_0,j)\} \\
\mathrm{lookuprpt}(l;i,j)\{\alpha,j_0\} \\
\{\mathrm{dlseg}\,\alpha\,(i,\mathbf{nil},j_0,j) \wedge l = j_0\}
\end{array}\right\} * \ \mathrm{dlseg}\,\beta\,(j_0,j,\mathbf{nil},m) \\
\{(\mathrm{dlseg}\,\alpha\,(i,\mathbf{nil},j_0,j) \ * \ \mathrm{dlseg}\,\beta\,(j_0,j,\mathbf{nil},m)) \wedge l = j_0\} \\
\{\mathrm{dlseg}\,\alpha\,(i,\mathbf{nil},l,j) \ * \ \mathrm{dlseg}\,\beta\,(l,j,\mathbf{nil},m)\}
\end{array}\right\} \exists j_0$$

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

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

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

$$\left.\begin{array}{l}
\{\mathrm{dlseg}\,\alpha\,(i,\mathbf{nil},l,j)\} \\
\mathrm{setrpt}(i;k,j)\{\alpha,l\} \\
\{\mathrm{dlseg}\,\alpha\,(i,\mathbf{nil},k,j)\}
\end{array}\right\} * \ k \mapsto a,l,j \ * \ \mathrm{dlseg}\,\beta\,(l,j,\mathbf{nil},m)$$

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

$$\left.\begin{array}{l}
\{\mathrm{dlseg}\,\beta\,(l,j,\mathbf{nil},m)\} \\
\mathrm{setlpt}(m;l,k)\{\beta,j\} \\
\{\mathrm{dlseg}\,\beta\,(l,k,\mathbf{nil},m)\}
\end{array}\right\} * \ \mathrm{dlseg}\,\alpha\,(i,\mathbf{nil},k,j) \ * \ k \mapsto a,l,j$$

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

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

—

Reasoning about the Call of lookuprpt

Beginning with the hypothesis

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

we use (GCALLan) to infer

$$\{\mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{j}_0, \mathsf{j})\}$$
$$\mathsf{lookuprpt}(\mathsf{l}; \mathsf{i}, \mathsf{j})\{\alpha, \mathsf{j}_0\}$$
$$\{\mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{j}_0, \mathsf{j}) \wedge \mathsf{l} = \mathsf{j}_0\}.$$

Next, we use (FRan), the purity of $\mathsf{l} = \mathsf{j}_0$, and obvious properties of equality to obtain

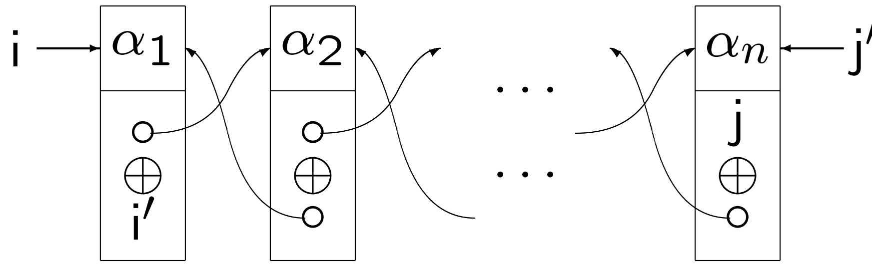
$$\{\mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{j}_0, \mathsf{j}) \,*\, \mathsf{dlseg}\,\beta\,(\mathsf{j}_0, \mathsf{j}, \mathbf{nil}, \mathsf{m})\}$$
$$\mathsf{lookuprpt}(\mathsf{l}; \mathsf{i}, \mathsf{j})\{\alpha, \mathsf{j}_0\}$$
$$\{(\mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{j}_0, \mathsf{j}) \wedge \mathsf{l} = \mathsf{j}_0) \,*\, \mathsf{dlseg}\,\beta\,(\mathsf{j}_0, \mathsf{j}, \mathbf{nil}, \mathsf{m})\}$$
$$\{(\mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{j}_0, \mathsf{j}) \,*\, \mathsf{dlseg}\,\beta\,(\mathsf{j}_0, \mathsf{j}, \mathbf{nil}, \mathsf{m})) \wedge \mathsf{l} = \mathsf{j}_0\}$$
$$\{\mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j}) \,*\, \mathsf{dlseg}\,\beta\,(\mathsf{l}, \mathsf{j}, \mathbf{nil}, \mathsf{m})\}.$$

Finally, we use (EQan), as well as predicate-calculus rules for renaming and eliminating existential quantifiers, to obtain

$$\{\exists \mathsf{l}.\ \mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j}) \,*\, \mathsf{dlseg}\,\beta\,(\mathsf{l}, \mathsf{j}, \mathbf{nil}, \mathsf{m})\}$$
$$\{\exists \mathsf{j}_0.\ \mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{j}_0, \mathsf{j}) \,*\, \mathsf{dlseg}\,\beta\,(\mathsf{j}_0, \mathsf{j}, \mathbf{nil}, \mathsf{m})\}$$
$$\mathsf{lookuprpt}(\mathsf{l}; \mathsf{i}, \mathsf{j})\{\alpha, \mathsf{j}_0\}$$
$$\{\exists \mathsf{j}_0.\ \mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j}) \,*\, \mathsf{dlseg}\,\beta\,(\mathsf{l}, \mathsf{j}, \mathbf{nil}, \mathsf{m})\}$$
$$\{\mathsf{dlseg}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j}) \,*\, \mathsf{dlseg}\,\beta\,(\mathsf{l}, \mathsf{j}, \mathbf{nil}, \mathsf{m})\}.$$

Xor-Linked List Segments

xlseg α $(\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}')$:



is defined by

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

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

—

Properties

$$\begin{aligned}
\mathsf{xlseg}\ \mathsf{a}\,(\mathsf{i},\mathsf{i}',\mathsf{j},\mathsf{j}') &\Leftrightarrow \mathsf{i} \mapsto \mathsf{a}, (\mathsf{j} \oplus \mathsf{i}') \wedge \mathsf{i} = \mathsf{j}' \\
\mathsf{xlseg}\ \alpha{\cdot}\beta\,(\mathsf{i},\mathsf{i}',\mathsf{k},\mathsf{k}') &\Leftrightarrow \exists \mathsf{j},\mathsf{j}'.\ \mathsf{xlseg}\ \alpha\,(\mathsf{i},\mathsf{i}',\mathsf{j},\mathsf{j}') \ *\ \mathsf{xlseg}\ \beta\,(\mathsf{j},\mathsf{j}',\mathsf{k},\mathsf{k}') \\
\mathsf{xlseg}\ \alpha{\cdot}\mathsf{b}\,(\mathsf{i},\mathsf{i}',\mathsf{k},\mathsf{k}') &\Leftrightarrow \exists \mathsf{j}'.\ \mathsf{xlseg}\ \alpha\,(\mathsf{i},\mathsf{i}',\mathsf{k}',\mathsf{j}') \ *\ \mathsf{k}' \mapsto \mathsf{b}, (\mathsf{k} \oplus \mathsf{j}') \\
\mathsf{xlist}\ \alpha\,(\mathsf{i},\mathsf{j}') &\stackrel{\mathsf{def}}{=} \mathsf{xlseg}\ \alpha\,(\mathsf{i},\mathbf{nil},\mathbf{nil},\mathsf{j}').
\end{aligned}$$

—

Emptyness Conditions (as with dlseg)

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

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

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

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

—

A Set-Pointer Procedure for Xor-linked Lists

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

$\{\mathsf{xlseg}\ \alpha\ (\mathsf{i},\mathbf{nil},\mathsf{j},\mathsf{j}')\}$

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

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

$\mathsf{i} := \mathsf{k}$

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

$\mathbf{else}$

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

$$
\left.\begin{array}{l}
\{\mathsf{j}' \mapsto \mathsf{b}, (\mathsf{j} \oplus \mathsf{k}')\} \\
\mathbf{newvar}\ \mathsf{x}\ \mathbf{in} \\
\quad (\mathsf{x} := [\mathsf{j}' + 1]\ ; \\
\quad \{\mathsf{x} = \mathsf{j} \oplus \mathsf{k}' \wedge \mathsf{j}' \mapsto \mathsf{b}, -\} \\
\quad [\mathsf{j}' + 1] := \mathsf{x} \oplus \mathsf{j} \oplus \mathsf{k} \\
\quad \{\mathsf{x} = \mathsf{j} \oplus \mathsf{k}' \wedge \mathsf{j}' \mapsto \mathsf{b}, \mathsf{x} \oplus \mathsf{j} \oplus \mathsf{k}\} \\
\quad \{\mathsf{j}' \mapsto \mathsf{b}, \mathsf{j} \oplus \mathsf{k}' \oplus \mathsf{j} \oplus \mathsf{k}\}) \\
\{\mathsf{j}' \mapsto \mathsf{b}, \mathsf{k} \oplus \mathsf{k}'\}
\end{array}\right\} * \left(\begin{array}{l}
\alpha = \alpha'{\cdot}\mathsf{b} \wedge \\
\mathsf{xlseg}\ \alpha' \\
\quad (\mathsf{i},\mathbf{nil},\mathsf{j}',\mathsf{k}')
\end{array}\right) \Bigg\}\ \exists\alpha', \mathsf{b}, \mathsf{k}'
$$

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

$\{\mathsf{xlseg}\ \alpha\ (\mathsf{i},\mathbf{nil},\mathsf{k},\mathsf{j}')\}.$

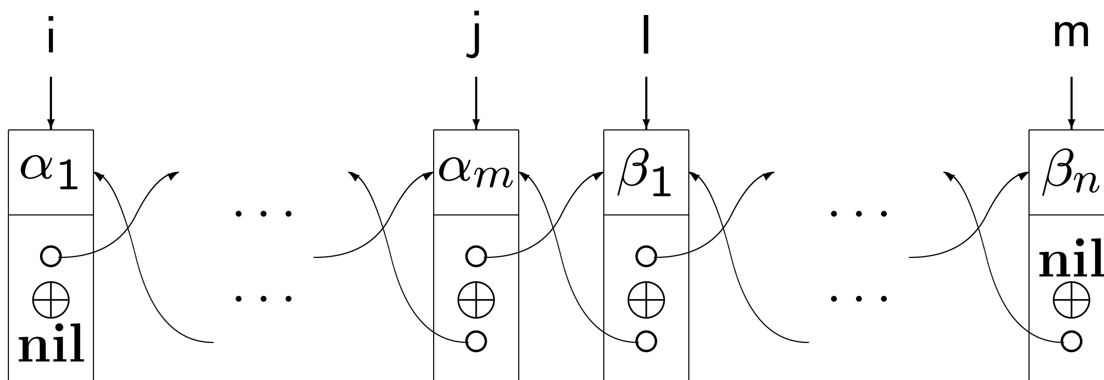
—

Upon Reflection

$$
\begin{array}{l}
\mathsf{xsetlpt}(\mathsf{j}'; \mathsf{i}, \mathsf{i}', \mathsf{k})\{\alpha\} = \\
\quad \{\mathsf{xlseg}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathbf{nil}, \mathsf{j}')\} \\
\quad \mathbf{if}\ \mathsf{i} = \mathbf{nil}\ \mathbf{then}\ \mathsf{j}' := \mathsf{k}\ \mathbf{else} \\
\qquad \mathbf{newvar}\ \mathsf{x}\ \mathbf{in}\ (\mathsf{x} := [\mathsf{i} + 1]\ ;\ [\mathsf{i} + 1] := \mathsf{x} \oplus \mathsf{i}' \oplus \mathsf{k}) \\
\quad \{\mathsf{xlseg}\ \alpha\ (\mathsf{i}, \mathsf{k}, \mathbf{nil}, \mathsf{j}')\}.
\end{array}
$$

—

Inserting an Element into a Xor-Linked List



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

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

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

$$\left.\begin{array}{l}\{\mathsf{xlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j})\} \\ \mathsf{xsetrpt}(\mathsf{i}; \mathsf{l}, \mathsf{j}, \mathsf{k})\{\alpha\} \\ \{\mathsf{xlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{k}, \mathsf{j})\}\end{array}\right\}\ *\ \mathsf{k} \mapsto \mathsf{a}, (\mathsf{l} \oplus \mathsf{j})\ *\ \mathsf{xlseg}\ \beta\ (\mathsf{l}, \mathsf{j}, \mathbf{nil}, \mathsf{m})$$

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

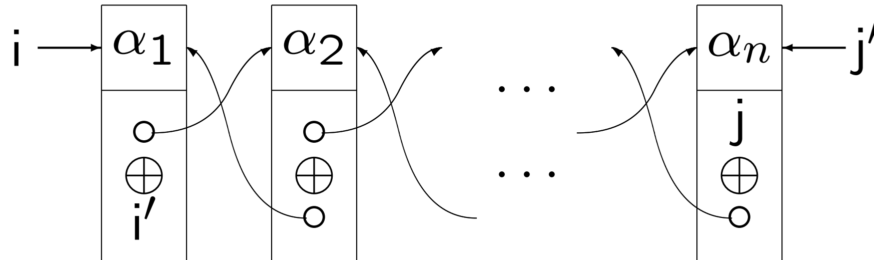
$$\left.\begin{array}{l}\{\mathsf{xlseg}\ \beta\ (\mathsf{l}, \mathsf{j}, \mathbf{nil}, \mathsf{m})\} \\ \mathsf{xsetlpt}(\mathsf{m}; \mathsf{l}, \mathsf{j}, \mathsf{k})\{\beta\} \\ \{\mathsf{xlseg}\ \beta\ (\mathsf{l}, \mathsf{k}, \mathbf{nil}, \mathsf{m})\}\end{array}\right\}\ *\ \mathsf{xlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{k}, \mathsf{j})\ *\ \mathsf{k} \mapsto \mathsf{a}, (\mathsf{l} \oplus \mathsf{j})$$

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

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

—

Reversing an Xor-Linked List Segment



Xor-linked segments have the extraordinary property that, as can be proved by induction on α,

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

and thus

$$\mathsf{xlist}\ \alpha\ (\mathsf{i}, \mathsf{j}') \Leftrightarrow \mathsf{xlist}\ \alpha^{\dagger}\ (\mathsf{j}', \mathsf{i}).$$

Thus xor-linked segments can be reversed in constant time, and xor-linked lists can be reversed without changing the heap.

—

Exercise 1

Write an annotated specification for a program that will remove an element from a cyclic buffer and assign it to y. The program should satisfy

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

$\cdots$

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

—

Exercise 2

Prove that $\exists \alpha.\ \mathsf{ntlseg}\ \alpha\ (\mathsf{i}, \mathsf{j})$ is a precise assertion.

—

Exercise 3

When

$$\exists\alpha, \beta.\ (\mathsf{lseg}\ \alpha\ (\mathsf{i}, \mathsf{j})\ *\ \mathsf{lseg}\ \beta\ (\mathsf{j}, \mathsf{k})) \wedge \gamma = \alpha{\cdot}\beta,$$

we say that j is an *interior pointer* of the list segment described by $\mathsf{lseg}\ \gamma\ (\mathsf{i}, \mathsf{k})$.

1. Give an assertion describing a list segment with two interior pointers j_1 and j_2, such that j_1 comes before than, or at the same point as, j_2 in the ordering of the elements of the list segment.
2. Give an assertion describing a list segment with two interior pointers j_1 and j_2, where there is no constraint on the relative positions of j_1 and j_2.
3. Prove that the first assertion implies the second.

—

Exercise 4

A *braced list segment* is a list segment with an interior pointer j to its last element; in the special case where the list segment is empty, j is $\mathbf{nil}$. Formally,

$$\mathsf{brlseg}\,\epsilon\,(\mathsf{i},\mathsf{j},\mathsf{k}) \stackrel{\text{def}}{=} \mathbf{emp} \wedge \mathsf{i} = \mathsf{k} \wedge \mathsf{j} = \mathbf{nil}$$

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

Prove the assertion

$$\mathsf{brlseg}\,\alpha\,(\mathsf{i},\mathsf{j},\mathsf{k}) \Rightarrow \mathbf{lseg}\,\alpha\,(\mathsf{i},\mathsf{k}).$$

—

Exercise 5

Write nonrecursive procedures for manipulating braced list segments, that satisfy the following hypotheses. In each case, give an annotated specification of the body that proves it is a correct implementation of the procedure. In a few cases, you may wish to use the procedures defined in previous cases.

1. A procedure for looking up the final pointer:

$$\{\mathsf{brlseg}\ \alpha\ (\mathsf{i}, \mathsf{j}, \mathsf{k}_0)\}\ \mathsf{lookuppt}(\mathsf{k}; \mathsf{i}, \mathsf{j})\{\alpha, \mathsf{k}_0\}$$
$$\{\mathsf{brlseg}\ \alpha\ (\mathsf{i}, \mathsf{j}, \mathsf{k}_0) \wedge \mathsf{k} = \mathsf{k}_0\}.$$

   (This procedure should not alter the heap.)

2. A procedure for setting the final pointer:

$$\{\mathsf{brlseg}\ \alpha\ (\mathsf{i}, \mathsf{j}, \mathsf{k}_0)\}\ \mathsf{setpt}(\mathsf{i}; \mathsf{j}, \mathsf{k})\{\alpha, \mathsf{k}_0\}\ \{\mathsf{brlseg}\ \alpha\ (\mathsf{i}, \mathsf{j}, \mathsf{k})\}.$$

   (This procedure should not allocate or deallocate heap storage.)

3. A procedure for appending an element on the left:

$$\{\mathsf{brlseg}\ \alpha\ (\mathsf{i}, \mathsf{j}, \mathsf{k}_0)\}\ \mathsf{appleft}(\mathsf{i}, \mathsf{j}; \mathsf{a})\{\alpha, \mathsf{k}_0\}\ \{\mathsf{brlseg}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{i}, \mathsf{j}, \mathsf{k}_0)\}.$$

4. A procedure for deleting an element on the left:

$$\{\mathsf{brlseg}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{i}, \mathsf{j}, \mathsf{k}_0)\}\ \mathsf{delleft}(\mathsf{i}, \mathsf{j};\ )\{\alpha, \mathsf{k}_0\}\ \{\mathsf{brlseg}\ \alpha\ (\mathsf{i}, \mathsf{j}, \mathsf{k}_0)\}.$$

5. A procedure for appending an element on the right:

$$\{\mathsf{brlseg}\ \alpha\ (\mathsf{i}, \mathsf{j}, \mathsf{k}_0)\}\ \mathsf{appright}(\mathsf{i}, \mathsf{j}; \mathsf{a})\{\alpha, \mathsf{k}_0\}\ \{\mathsf{brlseg}\ \alpha{\cdot}\mathsf{a}\ (\mathsf{i}, \mathsf{j}, \mathsf{k}_0)\}.$$

—

6. A procedure for concatenating two segments:

$$\{\mathsf{brlseg}\ \alpha\ (\mathsf{i}, \mathsf{j}, \mathsf{k}_0) * \mathsf{brlseg}\ \beta\ (\mathsf{i}', \mathsf{j}', \mathsf{k}'_0)\}$$
$$\mathsf{conc}(\mathsf{i}, \mathsf{j}; \mathsf{i}', \mathsf{j}')\{\alpha, \beta, \mathsf{k}_0, \mathsf{k}'_0\}$$
$$\{\mathsf{brlseg}\ \alpha{\cdot}\beta\ (\mathsf{i}, \mathsf{j}, \mathsf{k}'_0)\}.$$

(This procedure should not allocate or deallocate heap storage.)

—

Exercise 6

Rewrite the program for deleting an element from a doubly-linked list, to use the procedures setrpt and setlpt. Give an annotated specification. The program should satisfy:

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

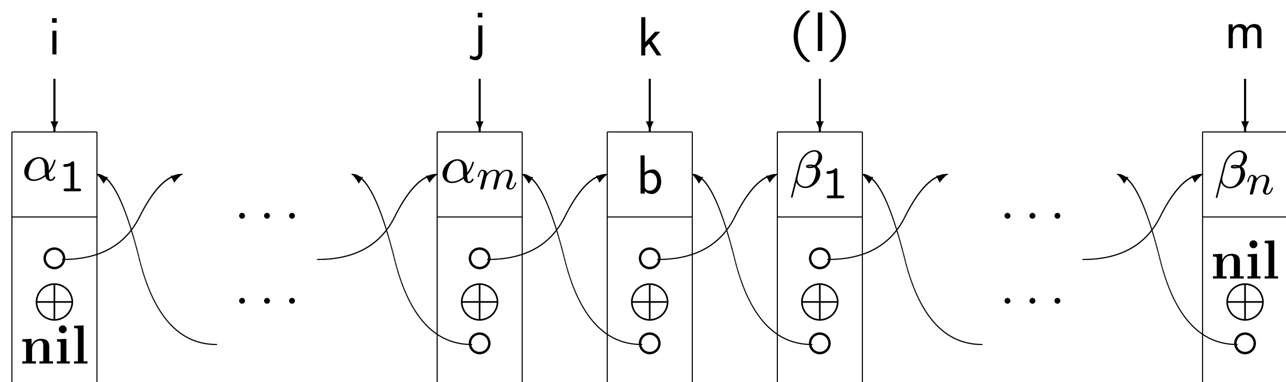
$$\cdots$$

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

—

Exercise 7

Give an annotated specification for a program to delete the element at k from an xor-linked list.



The program should use the procedures xsetrpt and xsetlpt, and should satisfy:

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

$\cdots$

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

—

Exercise 8

Prove, by structural induction on α, that

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

—