Class Notes for CS 818A4 - Spring 2004

SEPARATION LOGIC

4. Lists and List Segments

John C. Reynolds
Department of Computer Science
Carnegie Mellon University

Revised April 8, 2004

©2004 John C. Reynolds

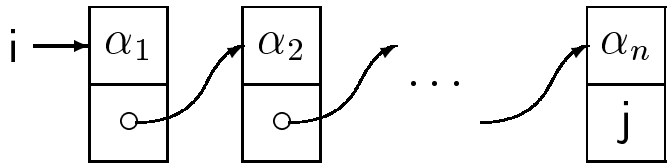
Notation for Sequences

When α and β are sequences, we write

- ϵ for the empty sequence.
- $[x]$ for the single-element sequence containing x. (We will omit the brackets when x is not a sequence.)
- $\alpha{\cdot}\beta$ for the composition of 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.

Singly-linked List Segments

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



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

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

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

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

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

Emptyness Conditions

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

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

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 $\mathsf{i}_0, \ldots, \mathsf{i}_{n-1}$ are distinct, but i_n is not constrained, and may equal any of the $\mathsf{i}_0, \ldots, \mathsf{i}_{n-1}$. In this case, we say that the list segment is *touching*.

We can define nontouching list segments inductively by:

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

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

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:

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

Inserting a List Element

- At the beginning:

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

- At the end of a nonempty segment:

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

Deleting a List Element

- At the beginning:

$$
\begin{array}{l}
\{\mathsf{lseg}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{i},\mathsf{k})\} \\
\{\exists \mathsf{j}.\ \mathsf{i} \mapsto \mathsf{a},\mathsf{j}\ *\ \mathsf{lseg}\ \alpha\ (\mathsf{j},\mathsf{k})\} \\
\{\exists \mathsf{j}.\ \mathsf{i}+1 \mapsto \mathsf{j}\ *\ (\mathsf{i} \mapsto \mathsf{a}\ *\ \mathsf{lseg}\ \alpha\ (\mathsf{j},\mathsf{k}))\} \\
\mathsf{j} := [\mathsf{i}+1]\ ; \\
\{\mathsf{i}+1 \mapsto \mathsf{j}\ *\ (\mathsf{i} \mapsto \mathsf{a}\ *\ \mathsf{lseg}\ \alpha\ (\mathsf{j},\mathsf{k}))\} \\
\mathbf{dispose}\ \mathsf{i}\ ; \\
\{\mathsf{i}+1 \mapsto \mathsf{j}\ *\ \mathsf{lseg}\ \alpha\ (\mathsf{j},\mathsf{k})\} \\
\mathbf{dispose}\ \mathsf{i}+1\ ; \\
\{\mathsf{lseg}\ \alpha\ (\mathsf{j},\mathsf{k})\} \\
\mathsf{i} := \mathsf{j} \\
\{\mathsf{lseg}\ \alpha\ (\mathsf{i},\mathsf{k})\}
\end{array}
$$

- At the end of a nonempty segment:

$$
\begin{array}{l}
\{\mathsf{lseg}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a},\mathsf{k}\ *\ \mathsf{k} \mapsto \mathsf{b},\mathsf{l}\} \\
[\mathsf{j}+1] := \mathsf{l}\ ; \\
\{\mathsf{lseg}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a},\mathsf{l}\ *\ \mathsf{k} \mapsto \mathsf{b},\mathsf{l}\} \\
\mathbf{dispose}\ \mathsf{k}\ ;\ \mathbf{dispose}\ \mathsf{k}+1 \\
\{\mathsf{lseg}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a},\mathsf{l}\} \\
\{\mathsf{lseg}\ \alpha{\cdot}\mathsf{a}\ (\mathsf{i},\mathsf{l})\}
\end{array}
$$

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 ($\#\alpha=0$) or full ($\#\beta=0$).

To insert an element:

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

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

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

$[\mathsf{j}]:=\mathsf{x}\ ;$

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

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

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

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

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

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

$\mathsf{m}:=\mathsf{m}+1$

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

A Cyclic Buffer (continued)

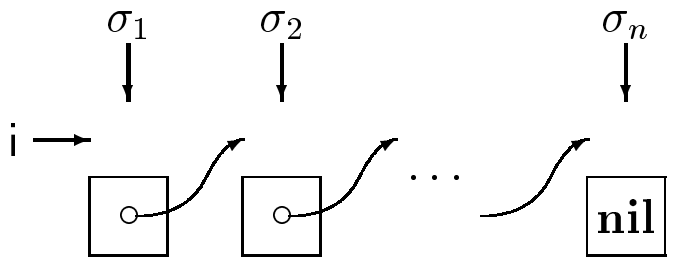
Note the above use of (LKG) for $j{:=}[j{+}1]$, with r as $((\mathsf{lseg}\,\alpha\,(\mathsf{i},\mathsf{j}')\ *\ \mathsf{j}'\mapsto\mathsf{x}\ *\ \mathsf{lseg}\,\beta\,(\mathsf{j}'',\mathsf{i}))\wedge\mathsf{m}=\#\alpha\wedge\mathsf{n}-1=\#\alpha+\#\beta)$.

To remove an element:

$\{\exists\beta.\ (\mathsf{lseg}\,\mathsf{a}{\cdot}\alpha\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{lseg}\,\beta\,(\mathsf{j},\mathsf{i}))\wedge$
$\quad\mathsf{m}=\#\mathsf{a}\cdot\alpha\wedge\mathsf{n}=\#\mathsf{a}{\cdot}\alpha+\#\beta\wedge\mathsf{m}>0\}$
$\{\exists\beta,\mathsf{i}''.\ (\mathsf{i}\mapsto\mathsf{a},\mathsf{i}''\ *\ \mathsf{lseg}\,\alpha\,(\mathsf{i}'',\mathsf{j})\ *\ \mathsf{lseg}\,\beta\,(\mathsf{j},\mathsf{i}))\wedge$
$\quad\mathsf{m}-1=\#\alpha\wedge\mathsf{n}-1=\#\alpha+\#\beta\}$
$\{\exists\beta,\mathsf{i}'',\mathsf{y}.\ (\mathsf{i}\mapsto\mathsf{y},\mathsf{i}''\ *\ \mathsf{lseg}\,\alpha\,(\mathsf{i}'',\mathsf{j})\ *\ \mathsf{lseg}\,\beta\,(\mathsf{j},\mathsf{i}))\wedge$
$\quad\mathsf{m}-1=\#\alpha\wedge\mathsf{n}-1=\#\alpha+\#\beta\wedge\mathsf{y}=\mathsf{a}\}$
$\mathsf{y}:=[\mathsf{i}]\ ;$
$\{\exists\beta,\mathsf{i}''.\ (\mathsf{i}\mapsto\mathsf{y},\mathsf{i}''\ *\ \mathsf{lseg}\,\alpha\,(\mathsf{i}'',\mathsf{j})\ *\ \mathsf{lseg}\,\beta\,(\mathsf{j},\mathsf{i}))\wedge$
$\quad\mathsf{m}-1=\#\alpha\wedge\mathsf{n}-1=\#\alpha+\#\beta\wedge\mathsf{y}=\mathsf{a}\}$
$\mathsf{i}:=[\mathsf{i}+1]\ ;$
$\{\exists\beta,\mathsf{i}'.\ (\mathsf{i}'\mapsto\mathsf{y},\mathsf{i}\ *\ \mathsf{lseg}\,\alpha\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{lseg}\,\beta\,(\mathsf{j},\mathsf{i}'))\wedge$
$\quad\mathsf{m}-1=\#\alpha\wedge\mathsf{n}-1=\#\alpha+\#\beta\wedge\mathsf{y}=\mathsf{a}\}$
$\{\exists\beta.\ (\mathsf{lseg}\,\alpha\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{lseg}\,\beta{\cdot}\mathsf{y}\,(\mathsf{j},\mathsf{i}))\wedge$
$\quad\mathsf{m}-1=\#\alpha\wedge\mathsf{n}=\#\alpha+\#\beta{\cdot}\mathsf{y}\wedge\mathsf{y}=\mathsf{a}\}$
$\{\exists\beta.\ (\mathsf{lseg}\,\alpha\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{lseg}\,\beta\,(\mathsf{j},\mathsf{i}))\wedge$
$\quad\mathsf{m}-1=\#\alpha\wedge\mathsf{n}=\#\alpha+\#\beta\wedge\mathsf{y}=\mathsf{a}\}$
$\mathsf{m}:=\mathsf{m}-1$
$\{\exists\beta.\ (\mathsf{lseg}\,\alpha\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{lseg}\,\beta\,(\mathsf{j},\mathsf{i}))\wedge\mathsf{m}=\#\alpha\wedge\mathsf{n}=\#\alpha+\#\beta\wedge\mathsf{y}=\mathsf{a}\}$

Bornat Lists

$\mathsf{listN}\ \sigma\ \mathsf{i}$:



is defined by

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

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

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

Reversing a Bornat List

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

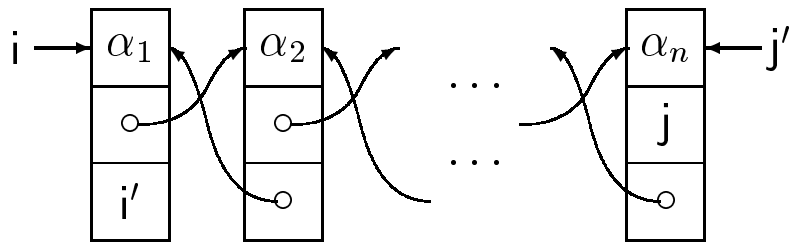
$\{\exists \sigma, \tau.\ (\mathsf{listN}\ \sigma\ \mathsf{i}\ *\ \mathsf{listN}\ \tau\ \mathsf{j}) \wedge \sigma_0^\dagger = \sigma^\dagger \cdot \tau\}\Big)$

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

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

Doubly-Linked List Segments

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



is defined by

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

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

Properties

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

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

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

Emptyness Conditions

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

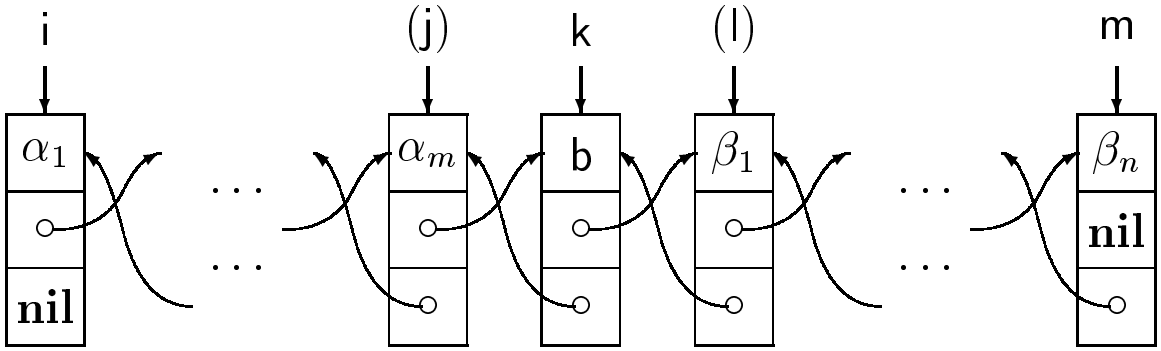
$$\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') \ \wedge \ \mathsf{j}' = \mathbf{nil} \ \Rightarrow \ (\alpha = \epsilon \wedge \mathsf{i}' = \mathbf{nil} \wedge \mathsf{i} = \mathsf{j})$$

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

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

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

Deleting an Element from a Doubly-Linked List



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

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

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

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

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

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

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

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

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

else

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

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

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

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

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

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

$\vdots$

Deleting from a Doubly-Linked List (continued)

$$
\begin{array}{l}
\quad \vdots \\
\{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j})\ *\ \mathsf{dlseg}\ \beta\ (\mathsf{l}, \mathsf{k}, \mathbf{nil}, \mathsf{m})\} \\
\mathbf{if}\ \mathsf{l} = \mathbf{nil}\ \mathbf{then} \\
\quad \{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j}) \wedge \mathsf{l} = \mathbf{nil} \wedge \mathsf{k} = \mathsf{m} \wedge \beta = \epsilon\} \\
\quad \mathsf{m} := \mathsf{j} \\
\quad \{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j}) \wedge \mathsf{l} = \mathbf{nil} \wedge \mathsf{j} = \mathsf{m} \wedge \beta = \epsilon\} \\
\mathbf{else} \\
\quad \{\exists \mathsf{a}, \beta', \mathsf{n}.\ (\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j})\ *\ \mathsf{l} \mapsto \mathsf{a}, \mathsf{n}, \mathsf{k} \\
\qquad *\ \mathsf{dlseg}\ \beta'\ (\mathsf{n}, \mathsf{l}, \mathbf{nil}, \mathsf{m})) \wedge \beta = \mathsf{a} \cdot \beta'\} \\
\quad [\mathsf{l} + 2] := \mathsf{j} \\
\quad \{\exists \mathsf{a}, \beta', \mathsf{n}.\ (\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j})\ *\ \mathsf{l} \mapsto \mathsf{a}, \mathsf{n}, \mathsf{j} \\
\qquad *\ \mathsf{dlseg}\ \beta'\ (\mathsf{n}, \mathsf{l}, \mathbf{nil}, \mathsf{m})) \wedge \beta = \mathsf{a} \cdot \beta'\} \\
\{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j})\ *\ \mathsf{dlseg}\ \beta\ (\mathsf{l}, \mathsf{j}, \mathbf{nil}, \mathsf{m})\} \\
\{\mathsf{dlseg}\ \alpha \cdot \beta\ (\mathsf{i}, \mathbf{nil}, \mathbf{nil}, \mathsf{m})\}
\end{array}
$$

Inference for Simple Procedures

A simple procedure definition has the form

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

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

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

Simple Procedures (SPROC)

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

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

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

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

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

Simple Recursive Procedures (SRPROC)

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

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

A Lookup-Pointer Procedure for Doubly-Linked Lists

Since

$\{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}_0, \mathsf{j}')\}$
$\mathbf{if}\ \mathsf{j}' = \mathbf{nil}\ \mathbf{then}$
$\quad \{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}_0, \mathsf{j}') \wedge \mathsf{i} = \mathsf{j}_0\}$
$\quad \mathsf{j} := \mathsf{i}$
$\mathbf{else}$
$\quad \{\exists \alpha', \mathsf{b}, \mathsf{k}'.\ \ \alpha = \alpha' \cdot \mathsf{b} \wedge (\mathsf{dlseg}\ \alpha'\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}', \mathsf{k}')\ *\ \mathsf{j}' \mapsto \mathsf{b}, \mathsf{j}_0, \mathsf{k}')\}$
$\quad \mathsf{j} := [\mathsf{j}' + 1]$
$\quad \{\exists \alpha', \mathsf{b}, \mathsf{k}'.\ \ \alpha = \alpha' \cdot \mathsf{b} \wedge$
$\qquad (\mathsf{dlseg}\ \alpha'\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}', \mathsf{k}')\ *\ \mathsf{j}' \mapsto \mathsf{b}, \mathsf{j}_0, \mathsf{k}') \wedge \mathsf{j} = \mathsf{j}_0\}$
$\{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}_0, \mathsf{j}') \wedge \mathsf{j} = \mathsf{j}_0\},$

the procedure

$\mathsf{lookuprpt}(\mathsf{i}, \mathsf{j}'; \mathsf{j}) =$
$\quad \mathbf{if}\ \mathsf{j}' = \mathbf{nil}\ \mathbf{then}\ \mathsf{j} := \mathsf{i}\ \mathbf{else}\ \mathsf{j} := [\mathsf{j}' + 1]$

satisfies

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

A Set-Pointer Procedure for Doubly-Linked Lists

Since

$\{\exists \mathsf{j}.\ \ \mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}, \mathsf{j}')\}$
$\mathbf{if}\ \mathsf{j}' = \mathbf{nil}\ \mathbf{then}$
$\quad \{\alpha = \epsilon \wedge \mathbf{emp} \wedge \mathsf{j}' = \mathbf{nil}\}$
$\quad \mathsf{i} := \mathsf{j}$
$\quad \{\alpha = \epsilon \wedge \mathbf{emp} \wedge \mathsf{j}' = \mathbf{nil} \wedge \mathsf{i} = \mathsf{j}\}$
$\mathbf{else}$
$\quad \{\exists \alpha', \mathsf{b}, \mathsf{j}, \mathsf{k}'.\ \ \alpha = \alpha' \cdot \mathsf{b} \wedge (\mathsf{dlseg}\ \alpha'\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}', \mathsf{k}')\ *\ \mathsf{j}' \mapsto \mathsf{b}, \mathsf{j}, \mathsf{k}')\}$
$\quad [\mathsf{j}' + 1] := \mathsf{j}$
$\quad \{\exists \alpha', \mathsf{b}, \mathsf{k}'.\ \ \alpha = \alpha' \cdot \mathsf{b} \wedge (\mathsf{dlseg}\ \alpha'\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}', \mathsf{k}')\ *\ \mathsf{j}' \mapsto \mathsf{b}, \mathsf{j}, \mathsf{k}')\}$
$\{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}, \mathsf{j}')\},$

the procedure

$\mathsf{setrpt}(\mathsf{j}, \mathsf{j}'; \mathsf{i}) =$
$\quad \mathbf{if}\ \mathsf{j}' = \mathbf{nil}\ \mathbf{then}\ \mathsf{i} := \mathsf{j}\ \mathbf{else}\ [\mathsf{j}' + 1] := \mathsf{j}$

satisfies

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

Upon Reflection

The procedure

$$\mathsf{lookup|pt}(\mathsf{i}, \mathsf{j}'; \mathsf{i}') = \\ \quad \textbf{if } \mathsf{i} = \textbf{nil } \textbf{then } \mathsf{i}' := \mathsf{j}' \textbf{ else } \mathsf{i}' := [\mathsf{i} + 2]$$

satisfies

$$\{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathsf{i}'_0, \textbf{nil}, \mathsf{j}')\} \\ \mathsf{lookup|pt}(\mathsf{i}, \mathsf{j}'; \mathsf{i}') \\ \{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathsf{i}'_0, \textbf{nil}, \mathsf{j}') \wedge \mathsf{i}' = \mathsf{i}'_0\},$$

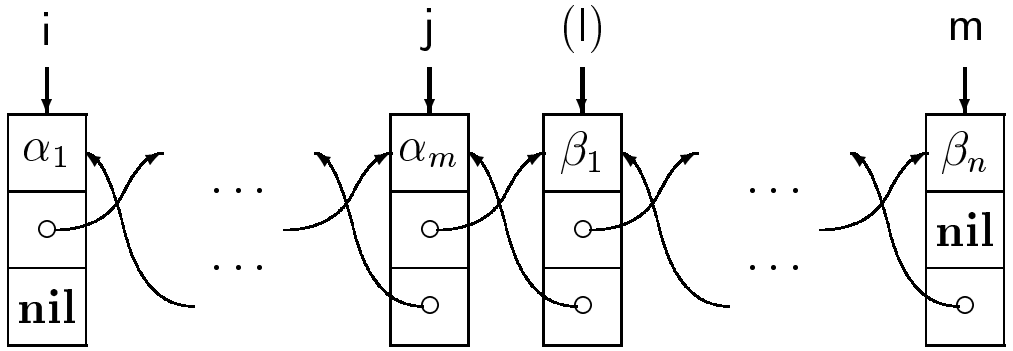
and the procedure

$$\mathsf{set|pt}(\mathsf{i}, \mathsf{i}'; \mathsf{j}') = \\ \quad \textbf{if } \mathsf{i} = \textbf{nil } \textbf{then } \mathsf{j}' := \mathsf{i}' \textbf{ else } [\mathsf{i} + 2] := \mathsf{i}'$$

satisfies

$$\{\exists \mathsf{i}'.\ \ \mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathsf{i}', \textbf{nil}, \mathsf{j}')\} \\ \mathsf{set|pt}(\mathsf{i}, \mathsf{i}'; \mathsf{j}') \\ \{\mathsf{dlseg}\ \alpha\ (\mathsf{i}, \mathsf{i}', \textbf{nil}, \mathsf{j}')\}.$$

Inserting an Element into a Doubly-Linked List



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

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

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

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

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

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

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

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

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

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

Deriving the Specification of the Call of lookuprpt

From:

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

using Free Variable Substitution (FVS):

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

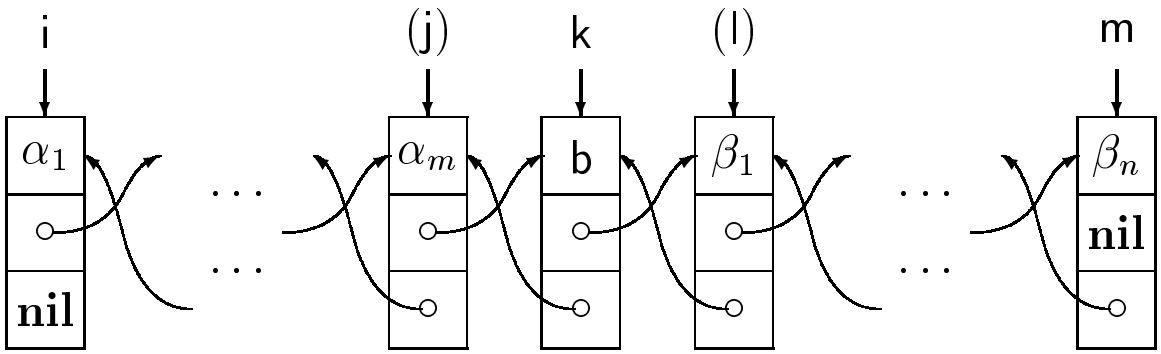
using the Frame Rule (FR) and an obvious property of equality:

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

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

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

Deleting an Element from a Doubly-Linked List



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

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

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

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

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

$\mathbf{setrpt}(\mathsf{l}, \mathsf{j}; \mathsf{i})\ ;$

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

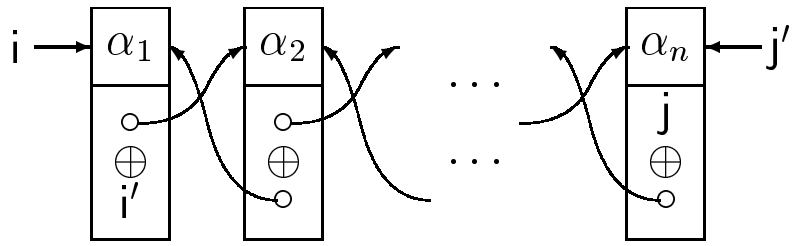
$\mathbf{setlpt}(\mathsf{l}, \mathsf{j}; \mathsf{m})$

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

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

An Inductive Definition: Xor-Linked List Segments

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



is defined by

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

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

Properties

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

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

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

Emptyness Conditions (as with dlseg)

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

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

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

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

A Set-Pointer Procedure for Xor-linked Lists

Since

$\{\mathsf{x}|\mathsf{seg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}, \mathsf{j}')\}$
$\mathbf{if}\ \mathsf{j}' = \mathbf{nil}\ \mathbf{then}$
$\{\alpha = \epsilon \wedge \mathbf{emp} \wedge \mathsf{j}' = \mathbf{nil}\}$
$\mathsf{i} := \mathsf{k}$
$\{\alpha = \epsilon \wedge \mathbf{emp} \wedge \mathsf{j}' = \mathbf{nil} \wedge \mathsf{i} = \mathsf{k}\}$
$\mathbf{else}$
$\{\exists \alpha', \mathsf{b}, \mathsf{k}'.\ \ \alpha = \alpha' \cdot \mathsf{b} \wedge (\mathsf{x}|\mathsf{seg}\ \alpha'\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}', \mathsf{k}')\ *\ \mathsf{j}' \mapsto \mathsf{b}, (\mathsf{j} \oplus \mathsf{k}'))\}$
$\mathbf{newvar}\ \mathsf{x}\ \mathbf{in}\ (\mathsf{x} := [\mathsf{j}' + 1]\ ;\ [\mathsf{j}' + 1] := \mathsf{x} \oplus \mathsf{j} \oplus \mathsf{k})$
$\{\exists \alpha', \mathsf{b}, \mathsf{k}'.\ \ \alpha = \alpha' \cdot \mathsf{b} \wedge (\mathsf{x}|\mathsf{seg}\ \alpha'\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}', \mathsf{k}')\ *\ \mathsf{j}' \mapsto \mathsf{b}, (\mathsf{k} \oplus \mathsf{k}'))\}$
$\{\mathsf{x}|\mathsf{seg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{k}, \mathsf{j}')\}$,

the procedure

$\mathsf{xsetrpt}(\mathsf{j}, \mathsf{j}', \mathsf{k}; \mathsf{i}) =$
$\mathbf{if}\ \mathsf{j}' = \mathbf{nil}\ \mathbf{then}\ \mathsf{i} := \mathsf{k}\ \mathbf{else}$
$\mathbf{newvar}\ \mathsf{x}\ \mathbf{in}\ (\mathsf{x} := [\mathsf{j}' + 1]\ ;\ [\mathsf{j}' + 1] := \mathsf{x} \oplus \mathsf{j} \oplus \mathsf{k})$

satisfies

$\{\mathsf{x}|\mathsf{seg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}, \mathsf{j}')\}$
$\mathsf{xsetrpt}(\mathsf{j}, \mathsf{j}', \mathsf{k}; \mathsf{i})$
$\{\mathsf{x}|\mathsf{seg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{k}, \mathsf{j}')\}$.

Upon Reflection

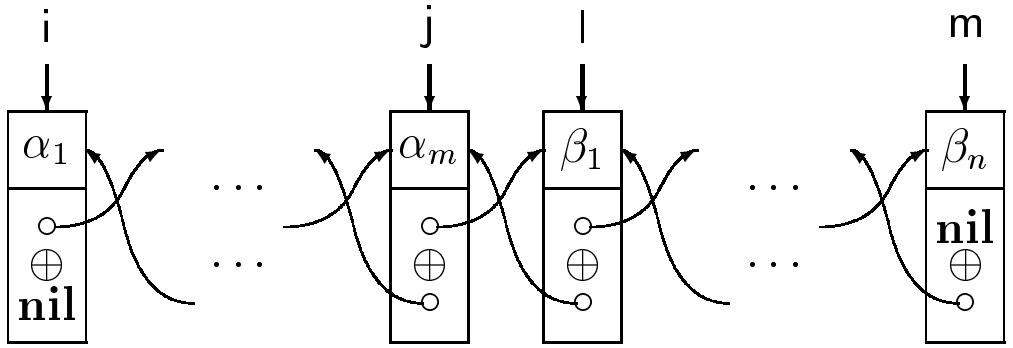
The procedure

$$\mathsf{xset|pt}(\mathsf{i}, \mathsf{i}', \mathsf{k}; \mathsf{j}') =$$
$$\quad \textbf{if } \mathsf{i} = \textbf{nil } \textbf{then } \mathsf{j}' := \mathsf{k} \textbf{ else}$$
$$\quad\quad \textbf{newvar } \mathsf{x} \textbf{ in } (\mathsf{x} := [\mathsf{i}+1] \,;\, [\mathsf{i}+1] := \mathsf{x} \oplus \mathsf{i}' \oplus \mathsf{k})$$

satisfies

$$\{\mathsf{x|seg}\ \alpha\ (\mathsf{i}, \mathsf{i}', \textbf{nil}, \mathsf{j}')\}$$
$$\mathsf{xset|pt}(\mathsf{i}, \mathsf{i}', \mathsf{k}; \mathsf{j}')$$
$$\{\mathsf{x|seg}\ \alpha\ (\mathsf{i}, \mathsf{k}, \textbf{nil}, \mathsf{j}')\}.$$

Inserting an Element into a Xor-Linked List



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

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

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

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

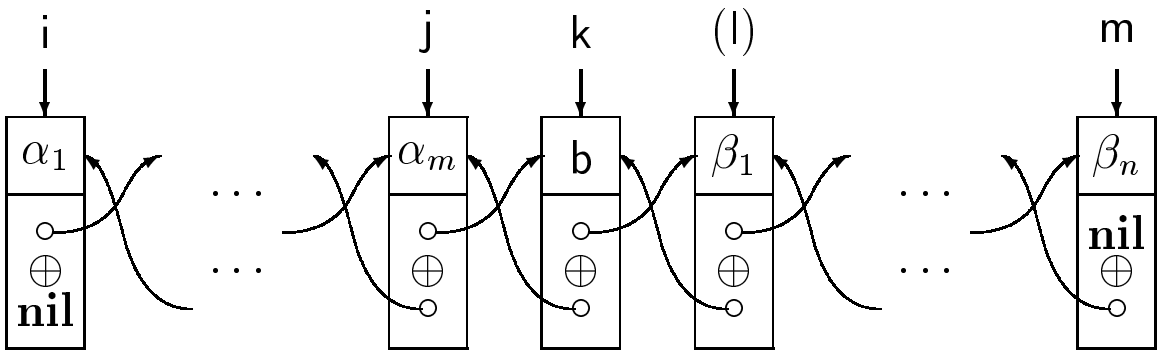
$\{\mathsf{xlseg}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{k}, \mathsf{j})\ *\ \mathsf{k} \mapsto \mathsf{a}, (\mathsf{l} \oplus \mathsf{j})\ *\ \mathsf{xlseg}\ \beta\ (\mathsf{l}, \mathsf{j}, \mathbf{nil}, \mathsf{m})\}$

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

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

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

Deleting an Element from a Xor-Linked List



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

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

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

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

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

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

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

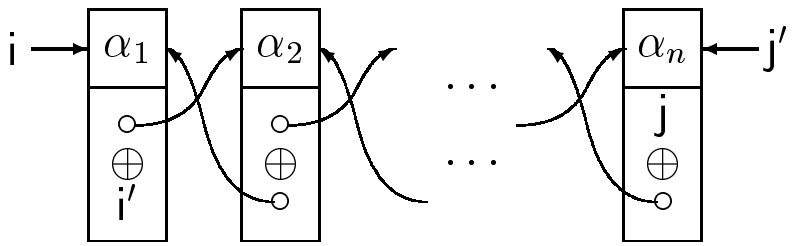
$\mathbf{xsetlpt}(\mathsf{l}, \mathsf{k}, \mathsf{j}; \mathsf{m})$

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

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

Reversing an Xor-Linked List Segment

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



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

Proof (by induction on α):

Base case:

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

Induction step:

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

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

$$\{\epsilon\} = \{\} \tag{2}$$

$$\{[x]\} = \{x\} \tag{3}$$

$$\{\alpha{\cdot}\beta\} = \{\alpha\} \cup \{\beta\} \tag{4}$$

$$\#\{\alpha\} \leq \#\alpha. \tag{5}$$

Pointwise Extension of Relations

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

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

Pointwise extension satisfies:

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

$$T' \subseteq T \wedge S\ \rho^*\ T \ \Rightarrow\ S\ \rho^*\ T' \tag{7}$$

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

$$S\ \rho^*\ \{\} \tag{9}$$

$$\{x\}\ \rho^*\ \{y\} \ \Leftrightarrow\ x\ \rho\ y \tag{10}$$

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

$$S\ \rho^*\ (T \cup T') \ \Leftrightarrow\ S\ \rho^*\ T \wedge S\ \rho^*\ T'. \tag{12}$$

Some Abbreviations

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

Ordering

We write **ord** α if the sequence α is ordered in nonstrict increasing order. Then

$$\#\alpha \leq 1 \quad \Rightarrow \quad \textbf{ord}\ \alpha \tag{13}$$

$$\textbf{ord}\ \alpha{\cdot}\beta \quad \Leftrightarrow \quad \textbf{ord}\ \alpha \wedge \textbf{ord}\ \beta \wedge \{\alpha\} \leq^* \{\beta\}. \tag{14}$$

Rearrangement

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

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

Then

$$\alpha \sim \alpha \tag{15}$$

$$\alpha \sim \beta \quad \Rightarrow \quad \beta \sim \alpha \tag{16}$$

$$\alpha \sim \beta \wedge \beta \sim \gamma \quad \Rightarrow \quad \alpha \sim \gamma \tag{17}$$

$$\alpha \sim \alpha' \wedge \beta \sim \beta' \quad \Rightarrow \quad \alpha{\cdot}\beta \sim \alpha'{\cdot}\beta' \tag{18}$$

$$\alpha{\cdot}\beta \sim \beta{\cdot}\alpha \tag{19}$$

$$\alpha \sim \beta \quad \Rightarrow \quad \{\alpha\} = \{\beta\}. \tag{20}$$

Specifications for Sorting by Merging

We define the abbreviation

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

Then we will define procedures satisfying

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

and

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

An Arithmetical Note

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

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

Thus if $\mathsf{n1} = \mathsf{n} \div 2$ and $\mathsf{n2} = \mathsf{n} - \mathsf{n1}$, then

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

A Proof for mergesort

$\{\mathsf{lseg}\,\alpha\,(\mathsf{i},\mathsf{j}_0) \wedge \#\alpha = \mathsf{n} \wedge \mathsf{n} \geq 1\}$
if $\mathsf{n} = 1$ **then**
  $\{\mathsf{lseg}\,\alpha\,(\mathsf{i},-) \wedge \mathbf{ord}\,\alpha \wedge \mathsf{i} \mapsto -,\mathsf{j}_0\}$
  $\mathsf{j} := [\mathsf{i}+1]$
  $\{\mathsf{lseg}\,\alpha\,(\mathsf{i},-) \wedge \mathbf{ord}\,\alpha \wedge \mathsf{j} = \mathsf{j}_0\}$
else
  $\big(\mathsf{n1} := \mathsf{n} \div 2\,;\, \mathsf{n2} := \mathsf{n} - \mathsf{n1}\,;\, \mathsf{i1} := \mathsf{i}\,;$
  $\{\exists\alpha_1,\alpha_2,\mathsf{i}_2.\ ((\mathsf{lseg}\,\alpha_1\,(\mathsf{i1},\mathsf{i}_2) \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1)$
    $*\ (\mathsf{lseg}\,\alpha_2\,(\mathsf{i}_2,\mathsf{j}_0) \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1)) \wedge \alpha = \alpha_1{\cdot}\alpha_2\}$
  $\mathsf{mergesort}(\mathsf{n1};\mathsf{i1},\mathsf{i2})\,;$
  $\{\exists\alpha_1,\alpha_2,\beta_1.\ ((\mathsf{lseg}\,\beta_1\,(\mathsf{i1},-) \wedge \beta_1 \sim \alpha_1 \wedge \mathbf{ord}\,\beta_1 \wedge \#\alpha_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1)$
    $*\ (\mathsf{lseg}\,\alpha_2\,(\mathsf{i2},\mathsf{j}_0) \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1)) \wedge \alpha = \alpha_1{\cdot}\alpha_2\}$
  $\mathsf{mergesort}(\mathsf{n2};\mathsf{i2},\mathsf{j})\,;$
  $\{\exists\alpha_1,\alpha_2,\beta_1,\beta_2.\ ((\mathsf{lseg}\,\beta_1\,(\mathsf{i1},-) \wedge \beta_1 \sim \alpha_1 \wedge \mathbf{ord}\,\beta_1 \wedge \#\alpha_1 = \mathsf{n1}$
    $\wedge\ \mathsf{n1} \geq 1)$
    $*\ (\mathsf{lseg}\,\beta_2\,(\mathsf{i2},-) \wedge \beta_2 \sim \alpha_2 \wedge \mathbf{ord}\,\beta_2 \wedge \#\alpha_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1))$
   $\wedge\ \alpha = \alpha_1{\cdot}\alpha_2 \wedge \mathsf{j} = \mathsf{j}_0\}$
  $\{\exists\beta_1,\beta_2.\ ((\mathsf{lseg}\,\beta_1\,(\mathsf{i1},-) \wedge \mathbf{ord}\,\beta_1 \wedge \#\beta_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1)$
 $*\ (\mathsf{lseg}\,\beta_2\,(\mathsf{i2},-) \wedge \mathbf{ord}\,\beta_2 \wedge \#\beta_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1)) \wedge \alpha \sim \beta_1{\cdot}\beta_2 \wedge \mathsf{j} = \mathsf{j}_0\}$
  $\mathsf{merge}(\mathsf{n1},\mathsf{n2},\mathsf{i1},\mathsf{i2};\mathsf{i})$
  $\{\exists\beta,\beta_1,\beta_2.\ \mathsf{lseg}\,\beta(\mathsf{i},-) \wedge \beta \sim \beta_1{\cdot}\beta_2 \wedge \mathbf{ord}\,\beta \wedge \alpha \sim \beta_1{\cdot}\beta_2 \wedge \mathsf{j} = \mathsf{j}_0\}\big)$
$\{\exists\beta.\ \mathsf{lseg}\,\beta\,(\mathsf{i},-) \wedge \beta \sim \alpha \wedge \mathbf{ord}\,\beta \wedge \mathsf{j} = \mathsf{j}_0\}.$

A Proof for mergesort (continued)

Thus we may define

$$
\begin{array}{l}
\mathsf{mergesort}(\mathsf{n};\mathsf{i},\mathsf{j}) = \\
\quad \textbf{if } \mathsf{n} = 1 \textbf{ then } \mathsf{j} := [\mathsf{i}+1] \textbf{ else} \\
\quad\quad \Big(\mathsf{n1} := \mathsf{n} \div 2 \,;\, \mathsf{n2} := \mathsf{n} - \mathsf{n1} \,;\, \mathsf{i1} := \mathsf{i} \,; \\
\quad\quad \mathsf{mergesort}(\mathsf{n1};\mathsf{i1},\mathsf{i2}) \,;\, \mathsf{mergesort}(\mathsf{n2};\mathsf{i2},\mathsf{j}) \,; \\
\quad\quad \mathsf{merge}(\mathsf{n1},\mathsf{n2},\mathsf{i1},\mathsf{i2};\mathsf{i})\Big).
\end{array}
$$

A Proof for merge

Suppose

$$\{\exists \beta, \mathsf{a1}, \mathsf{j1}, \gamma_1, \mathsf{j2}, \gamma_2.$$
$$(\mathsf{lseg}\ \beta\ (\mathsf{i}, \mathsf{i1})\ *\ \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1}\ *\ \mathsf{lseg}\ \gamma_1\ (\mathsf{j1}, -)\ *\ \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2}$$
$$*\ \mathsf{lseg}\ \gamma_2\ (\mathsf{j2}, -))$$
$$\wedge\ \#\gamma_1 = \mathsf{n1} \wedge \#\gamma_2 = \mathsf{n2} \wedge \mathsf{a1} \leq \mathsf{a2} \wedge \beta{\cdot}\mathsf{a1}{\cdot}\gamma_1{\cdot}\mathsf{a2}{\cdot}\gamma_2 \sim \beta_1{\cdot}\beta_2$$
$$\wedge\ \mathbf{ord}\ (\mathsf{a1}{\cdot}\gamma_1) \wedge \mathbf{ord}\ (\mathsf{a2}{\cdot}\gamma_2) \wedge \mathbf{ord}\ \beta \wedge \{\beta\} \leq^* \{\mathsf{a1}{\cdot}\gamma_1\} \cup \{\mathsf{a2}{\cdot}\gamma_2\}\}$$
$$\mathsf{merge1}(\mathsf{n1}, \mathsf{n2}, \mathsf{i1}, \mathsf{i2}, \mathsf{a2})$$
$$\{\exists \beta.\ \mathsf{lseg}\ \beta\ (\mathsf{i}, -) \wedge \beta \sim \beta_1{\cdot}\beta_2 \wedge \mathbf{ord}\ \beta\}$$

Then

$$\{(\mathsf{lseg}\ \beta_1\ (\mathsf{i1}, -) \wedge \mathbf{ord}\ \beta_1 \wedge \#\beta_1 = \mathsf{n1} \wedge \mathsf{n1} \geq 1)$$
$$*\ (\mathsf{lseg}\ \beta_2\ (\mathsf{i2}, -) \wedge \mathbf{ord}\ \beta_2 \wedge \#\beta_2 = \mathsf{n2} \wedge \mathsf{n2} \geq 1)\}$$
$$\mathbf{newvar}\ \mathsf{a1}\ \mathbf{in}\ \mathbf{newvar}\ \mathsf{a2}\ \mathbf{in}\ \big(\mathsf{a1} := [\mathsf{i1}]\ ;\ \mathsf{a2} := [\mathsf{i2}]\ ;$$
$$\mathbf{if}\ \mathsf{a1} \leq \mathsf{a2}\ \mathbf{then}\ (\mathsf{i} := \mathsf{i1}\ ;\ \mathsf{merge1}(\mathsf{n1}, \mathsf{n2}, \mathsf{i1}, \mathsf{i2}, \mathsf{a2}))$$
$$\mathbf{else}\ (\mathsf{i} := \mathsf{i2}\ ;\ \mathsf{merge1}(\mathsf{n2}, \mathsf{n1}, \mathsf{i2}, \mathsf{i1}, \mathsf{a1}))\big)$$
$$\{\exists \beta.\ \mathsf{lseg}\ \beta\ (\mathsf{i}, -) \wedge \beta \sim \beta_1{\cdot}\beta_2 \wedge \mathbf{ord}\ \beta\}.$$

Thus we may define

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

A Proof for merge1

$$
\begin{array}{l}
\{\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}, -) \;*\; \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2} \\
\qquad *\; \mathsf{lseg}\,\gamma_2\,(\mathsf{j2}, -)) \\
\quad \wedge \#\gamma_1 = \mathsf{n1} \wedge \#\gamma_2 = \mathsf{n2} \wedge \mathsf{a1} \leq \mathsf{a2} \wedge \beta{\cdot}\mathsf{a1}{\cdot}\gamma_1{\cdot}\mathsf{a2}{\cdot}\gamma_2 \sim \beta_1{\cdot}\beta_2 \\
\quad \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\}\} \\
\mathbf{if}\ \mathsf{n1} = 0\ \mathbf{then}\ [\mathsf{i1}+1] := \mathsf{i2} \\
\quad \{\exists \beta, \mathsf{a1}, \mathsf{j2}, \gamma_2. \\
\qquad (\mathsf{lseg}\,\beta\,(\mathsf{i}, \mathsf{i1}) \;*\; \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{i2} \;*\; \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2} \;*\; \mathsf{lseg}\,\gamma_2\,(\mathsf{j2}, -)) \\
\qquad \wedge \#\gamma_2 = \mathsf{n2} \wedge \mathsf{a1} \leq \mathsf{a2} \wedge \beta{\cdot}\mathsf{a1}{\cdot}\mathsf{a2}{\cdot}\gamma_2 \sim \beta_1{\cdot}\beta_2 \\
\qquad \wedge \mathbf{ord}\,(\mathsf{a2}{\cdot}\gamma_2) \wedge \mathbf{ord}\,\beta \wedge \{\beta\} \leq^* \{\mathsf{a1}\} \cup \{\mathsf{a2}{\cdot}\gamma_2\}\} \\
\mathbf{else}\ \mathbf{newvar}\ \mathsf{j1}\ \mathbf{in}\ \mathbf{newvar}\ \mathsf{a3}\ \mathbf{in} \\
\quad \big(\mathsf{j1} := [\mathsf{i1}+1]\,;\, \mathsf{a3} := [\mathsf{j1}]\,; \\
\quad \{\exists \beta, \mathsf{a1}, \mathsf{j3}, \gamma_3, \mathsf{j2}, \gamma_2. \\
\qquad (\mathsf{lseg}\,\beta\,(\mathsf{i}, \mathsf{i1}) \;*\; \mathsf{i1} \mapsto \mathsf{a1}, \mathsf{j1} \;*\; \mathsf{j1} \mapsto \mathsf{a3}, \mathsf{j3} \;*\; \mathsf{lseg}\,\gamma_3\,(\mathsf{j3}, -) \\
\qquad\quad *\; \mathsf{i2} \mapsto \mathsf{a2}, \mathsf{j2} \;*\; \mathsf{lseg}\,\gamma_2\,(\mathsf{j2}, -)) \\
\quad \wedge \#\gamma_3 = \mathsf{n1} - 1 \wedge \#\gamma_2 = \mathsf{n2} \wedge \mathsf{a1} \leq \mathsf{a2} \wedge \beta{\cdot}\mathsf{a1}{\cdot}\mathsf{a3}{\cdot}\gamma_3{\cdot}\mathsf{a2}{\cdot}\gamma_2 \sim \beta_1{\cdot}\beta_2 \\
\quad \wedge \mathbf{ord}\,(\mathsf{a1}{\cdot}\mathsf{a3}{\cdot}\gamma_3) \wedge \mathbf{ord}\,(\mathsf{a2}{\cdot}\gamma_2) \wedge \mathbf{ord}\,\beta \\
\quad \wedge \{\beta\} \leq^* \{\mathsf{a1}{\cdot}\mathsf{a3}{\cdot}\gamma_3\} \cup \{\mathsf{a2}{\cdot}\gamma_2\}\} \\
\quad \mathbf{if}\ \mathsf{a3} \leq \mathsf{a2}\ \mathbf{then}\ \mathsf{merge1}(\mathsf{n1}-1, \mathsf{n2}, \mathsf{j1}, \mathsf{i2}, \mathsf{a2}) \\
\quad \mathbf{else}\ ([\mathsf{i1}+1] := \mathsf{i2}\,;\, \mathsf{merge1}(\mathsf{n2}, \mathsf{n1}-1, \mathsf{i2}, \mathsf{j1}, \mathsf{a3}))\big) \\
\{\exists \beta.\ \mathsf{lseg}\,\beta\,(\mathsf{i}, -) \wedge \beta \sim \beta_1{\cdot}\beta_2 \wedge \mathbf{ord}\,\beta\}
\end{array}
$$

A Proof for merge1 (continued)

Thus we may define

$$
\begin{array}{l}
\mathsf{merge1}(\mathsf{n1}, \mathsf{n2}, \mathsf{i1}, \mathsf{i2}, \mathsf{a2}) = \\
\quad \textbf{if } \mathsf{n1} = 0 \textbf{ then } [\mathsf{i1}+1] := \mathsf{i2} \\
\quad \textbf{else newvar } \mathsf{j1} \textbf{ in newvar } \mathsf{a3} \textbf{ in} \\
\quad\quad \big(\mathsf{j1} := [\mathsf{i1}+1] \,;\, \mathsf{a3} := [\mathsf{j1}] \,; \\
\quad\quad \textbf{if } \mathsf{a3} \leq \mathsf{a2} \textbf{ then } \mathsf{merge1}(\mathsf{n1}-1, \mathsf{n2}, \mathsf{j1}, \mathsf{i2}, \mathsf{a2}) \\
\quad\quad \textbf{else } ([\mathsf{i1}+1] := \mathsf{i2} \,;\, \mathsf{merge1}(\mathsf{n2}, \mathsf{n1}-1, \mathsf{i2}, \mathsf{j1}, \mathsf{a3}))\big)
\end{array}
$$