Class Notes for CS 819A4 - Spring 2003

4. Lists

John C. Reynolds
Department of Computer Science
Carnegie Mellon University

Revised May 2, 2003

©2003 John C. Reynolds

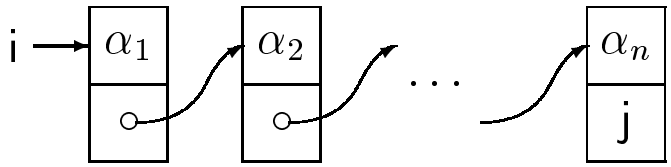
Notation for Sequences

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 Lists

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



is defined by

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

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

Properties

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

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

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

Proof of the Composition Property

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

The proof is by induction on the length of α.

When α is empty:

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

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

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

Cycles

When

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

we have

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

Thus $\mathsf{i}_0, \ldots, \mathsf{i}_{n-1}$ are distinct, but i_n is not constrained, so that $\mathsf{list}\ a_1 \cdots a_n\ (\mathsf{i}, \mathsf{i})$ may hold for any $n \geq 0$.

The obvious properties

$$\begin{array}{l}\mathsf{list}\ \alpha\ (\mathsf{i}, \mathsf{j}) \Rightarrow (\mathsf{i} = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge \mathsf{j} = \mathbf{nil})) \\ \mathsf{list}\ \alpha\ (\mathsf{i}, \mathsf{j}) \Rightarrow (\mathsf{i} \neq \mathsf{j} \Rightarrow \alpha \neq \epsilon)\end{array}$$

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

$$\begin{array}{c}\mathsf{list}\ \alpha\ (\mathsf{i}, \mathsf{j})\ *\ \mathsf{j} \hookrightarrow - \Rightarrow (\mathsf{i} = \mathsf{j} \Leftrightarrow \alpha = \epsilon) \\ \mathsf{list}\ \alpha\ (\mathsf{i}, \mathsf{j})\ *\ \mathsf{list}\ \beta\ (\mathsf{j}, \mathbf{nil}) \Rightarrow (\mathsf{i} = \mathsf{j} \Leftrightarrow \alpha = \epsilon).\end{array}$$

On the other hand, for example, a cyclic buffer is described by

$$\mathsf{list}\ \alpha\ (\mathsf{i}, \mathsf{j})\ *\ \mathsf{list}\ \beta\ (\mathsf{j}, \mathsf{i}).$$

Here, the buffer may be either empty or full when $\mathsf{i} = \mathsf{j}$.

Inserting a List Element

- At the beginning:

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

- At the end of a nonempty segment:

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

Deleting a List Element

- At the beginning:

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

- At the end of a nonempty segment:

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

Reversing a List

$\{\mathsf{list}\ \alpha_0\ (\mathsf{i}, \mathbf{nil})\}$

$\{\mathsf{list}\ \alpha_0\ (\mathsf{i}, \mathbf{nil})\ *\ (\mathbf{emp} \wedge \mathbf{nil} = \mathbf{nil})\}$

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

$\{\mathsf{list}\ \alpha_0\ (\mathsf{i}, \mathbf{nil})\ *\ (\mathbf{emp} \wedge \mathsf{j} = \mathbf{nil})\}$

$\{\mathsf{list}\ \alpha_0\ (\mathsf{i}, \mathbf{nil})\ *\ \mathsf{list}\ \epsilon\ (\mathsf{j}, \mathbf{nil})\}$

$\{\exists \alpha, \beta.\ (\mathsf{list}\ \alpha\ (\mathsf{i}, \mathbf{nil})\ *\ \mathsf{list}\ \beta\ (\mathsf{j}, \mathbf{nil})) \wedge \alpha_0^\dagger = \alpha^\dagger \cdot \beta\}$

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

$\Big(\{\exists \mathsf{a}, \alpha, \beta.\ (\mathsf{list}\ \mathsf{a}\cdot\alpha\ (\mathsf{i}, \mathbf{nil})\ *\ \mathsf{list}\ \beta\ (\mathsf{j}, \mathbf{nil})) \wedge \alpha_0^\dagger = (\mathsf{a}\cdot\alpha)^\dagger \cdot \beta\}$

$\{\exists \mathsf{a}, \alpha, \beta, \mathsf{k}.\ (\mathsf{i} \mapsto \mathsf{a}, \mathsf{k}\ *\ \mathsf{list}\ \alpha\ (\mathsf{k}, \mathbf{nil})\ *\ \mathsf{list}\ \beta\ (\mathsf{j}, \mathbf{nil}))$
$\wedge\ \alpha_0^\dagger = (\mathsf{a}\cdot\alpha)^\dagger \cdot \beta\}$

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

$\{\exists \mathsf{a}, \alpha, \beta.\ (\mathsf{i} \mapsto \mathsf{a}, \mathsf{k}\ *\ \mathsf{list}\ \alpha\ (\mathsf{k}, \mathbf{nil})\ *\ \mathsf{list}\ \beta\ (\mathsf{j}, \mathbf{nil}))$
$\wedge\ \alpha_0^\dagger = (\mathsf{a}\cdot\alpha)^\dagger \cdot \beta\}$

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

$\{\exists \mathsf{a}, \alpha, \beta.\ (\mathsf{i} \mapsto \mathsf{a}, \mathsf{j}\ *\ \mathsf{list}\ \alpha\ (\mathsf{k}, \mathbf{nil})\ *\ \mathsf{list}\ \beta\ (\mathsf{j}, \mathbf{nil}))$
$\wedge\ \alpha_0^\dagger = (\mathsf{a}\cdot\alpha)^\dagger \cdot \beta\}$

$\{\exists \mathsf{a}, \alpha, \beta.\ (\mathsf{list}\ \alpha\ (\mathsf{k}, \mathbf{nil})\ *\ \mathsf{list}\ \mathsf{a}\cdot\beta\ (\mathsf{i}, \mathbf{nil})) \wedge \alpha_0^\dagger = \alpha^\dagger \cdot \mathsf{a} \cdot \beta\}$

$\{\exists \alpha, \beta.\ (\mathsf{list}\ \alpha\ (\mathsf{k}, \mathbf{nil})\ *\ \mathsf{list}\ \beta\ (\mathsf{i}, \mathbf{nil})) \wedge \alpha_0^\dagger = \alpha^\dagger \cdot \beta\}$

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

$\{\exists \alpha, \beta.\ (\mathsf{list}\ \alpha\ (\mathsf{i}, \mathbf{nil})\ *\ \mathsf{list}\ \beta\ (\mathsf{j}, \mathbf{nil})) \wedge \alpha_0^\dagger = \alpha^\dagger \cdot \beta\}\Big)$

$\{\exists \alpha, \beta.\ \mathsf{list}\ \beta\ (\mathsf{j}, \mathbf{nil}) \wedge \alpha_0^\dagger = \alpha^\dagger \cdot \beta \wedge \alpha = \epsilon\}$

$\{\mathsf{list}\ \alpha_0^\dagger\ (\mathsf{j}, \mathbf{nil})\}$

A Cyclic Buffer

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

(If the buffer is empty ($\#\alpha=0$) or full ($\#\beta=0$), then $\mathsf{i}=\mathsf{j}$.)

To insert an element:

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

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

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

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

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

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

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

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

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

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

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

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

A Cyclic Buffer (continued)

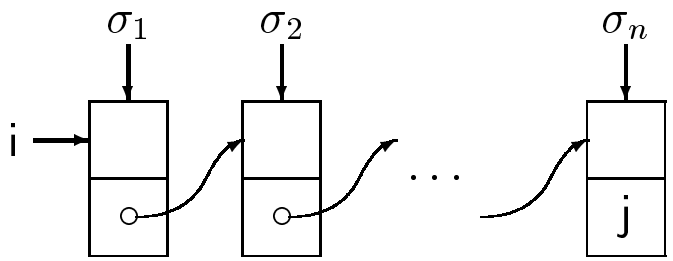
Note the use of (LKLG) with r taken as $((\mathsf{list}\,\alpha\,(\mathsf{i},\mathsf{j}')\ *\ \mathsf{j}'\mapsto\mathsf{x}\ *\ \mathsf{list}\,\beta\,(\mathsf{j}'',\mathsf{i}))\wedge\mathsf{m}=\#\alpha\wedge\mathsf{n}-1=\#\alpha+\#\beta)$.

To remove an element:

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

Bornat's Lists

$\mathsf{listN}\ \sigma\ (\mathsf{i}, \mathsf{j})$:



is defined by

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

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

Properties

$$\mathsf{listN}\ \mathsf{a}\ (\mathsf{i}, \mathsf{j}) \Leftrightarrow \mathsf{a} = \mathsf{i} \wedge \mathsf{i} + 1 \mapsto \mathsf{j}$$

$$\mathsf{listN}\ \sigma{\cdot}\tau\ (\mathsf{i}, \mathsf{k}) \Leftrightarrow \exists \mathsf{j}.\ \mathsf{listN}\ \sigma\ (\mathsf{i}, \mathsf{j})\ *\ \mathsf{listN}\ \tau\ (\mathsf{j}, \mathsf{k})$$

$$\mathsf{listN}\ \sigma{\cdot}\mathsf{b}\ (\mathsf{i}, \mathsf{k}) \Leftrightarrow \exists \mathsf{j}.\ \mathsf{listN}\ \sigma\ (\mathsf{i}, \mathsf{j})\ *\ (\mathsf{b} = \mathsf{k} \wedge \mathsf{j} + 1 \mapsto \mathsf{k})$$

Emptyness Conditions

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

$$\mathsf{listN}\ \sigma\ (\mathsf{i}, \mathsf{j}) \Rightarrow (\mathsf{i} \neq \mathsf{j} \Rightarrow \sigma \neq \epsilon)$$

$$\mathsf{listN}\ \sigma\ (\mathsf{i}, \mathsf{j})\ *\ \mathsf{j} + 1 \hookrightarrow - \Rightarrow (\mathsf{i} = \mathsf{j} \Leftrightarrow \sigma = \epsilon)$$

$$\mathsf{listN}\ \sigma\ (\mathsf{i}, \mathsf{j})\ *\ \mathsf{listN}\ \tau\ (\mathsf{j}, \mathbf{nil}) \Rightarrow (\mathsf{i} = \mathsf{j} \Leftrightarrow \sigma = \epsilon).$$

Reversing a Bornat List

$\{\mathsf{listN}\ \sigma_0\ (\mathsf{i}, \mathbf{nil})\}$

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

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

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

$\{\mathsf{listN}\ \sigma_0\ (\mathsf{i}, \mathbf{nil})\ *\ \mathsf{listN}\ \epsilon\ (\mathsf{j}, \mathbf{nil})\}$

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

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

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

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

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

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

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

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

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

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

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

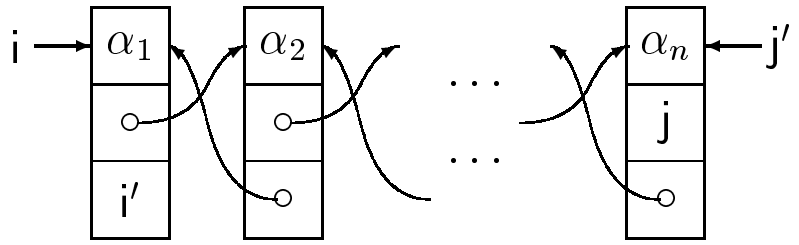
$\{\exists \sigma, \tau.\ (\mathsf{listN}\ \sigma\ (\mathsf{i}, \mathbf{nil})\ *\ \mathsf{listN}\ \tau\ (\mathsf{j}, \mathbf{nil})) \wedge \sigma_0^\dagger = \sigma^\dagger \cdot \tau\}\Big)$

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

$\{\mathsf{listN}\ \sigma_0^\dagger\ (\mathsf{j}, \mathbf{nil})\}$

Doubly-Linked Lists

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



is defined by

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

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

Properties

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

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

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

Emptyness Conditions

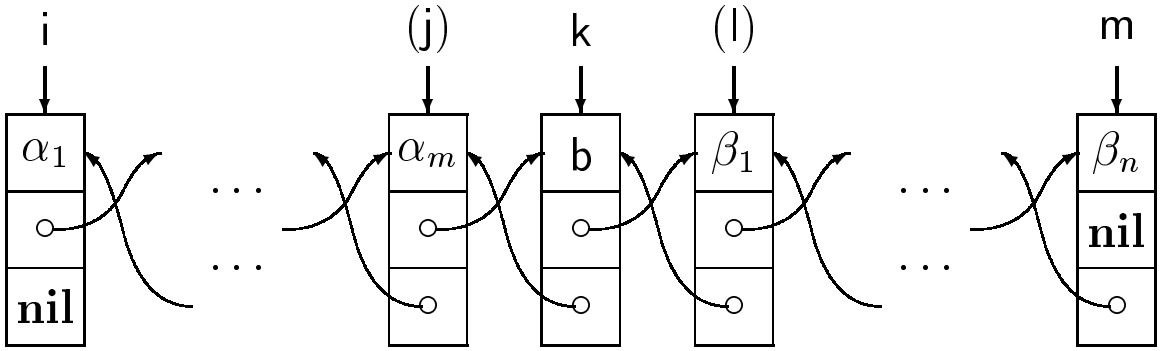
$$\mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') \ \wedge \quad \mathsf{i} = \mathbf{nil} \quad \Rightarrow \quad (\alpha = \epsilon \wedge \mathsf{j} = \mathbf{nil} \wedge \mathsf{i}' = \mathsf{j}')$$

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

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

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

Deleting an Element from a Doubly-Linked List



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

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

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

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

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

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

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

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

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

else

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

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

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

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

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

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

  $\vdots$

Deleting from a Doubly-Linked List (continued)

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

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 assigned 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

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

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

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

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

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

A Set-Pointer Procedure for Doubly-linked Lists

Since

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

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

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

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

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

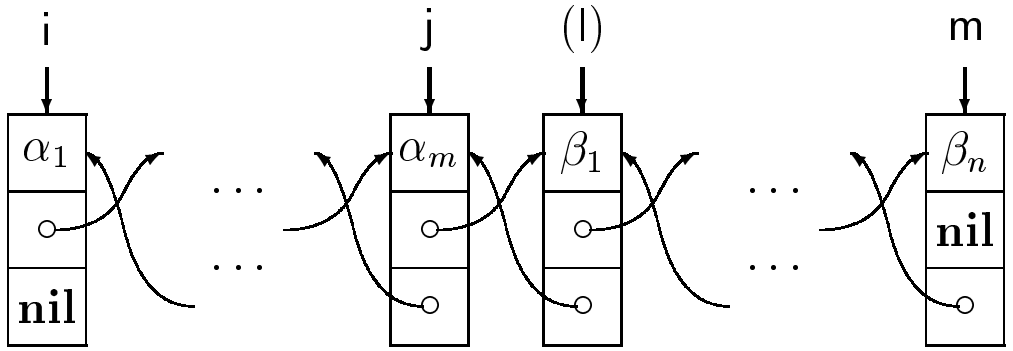
and the procedure

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

satisfies

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

Inserting an Element into a Doubly-Linked List



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

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

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

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

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

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

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

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

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

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

Deriving the Specification of the Call of lookuprpt

From:

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

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

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

using Free Variable Substitution (FVS):

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

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

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

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

$\{\mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}_0, \mathsf{j})\ *\ \mathsf{dlist}\ \beta\ (\mathsf{j}_0, \mathsf{j}, \mathbf{nil}, \mathsf{m})\}$

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

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

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

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

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

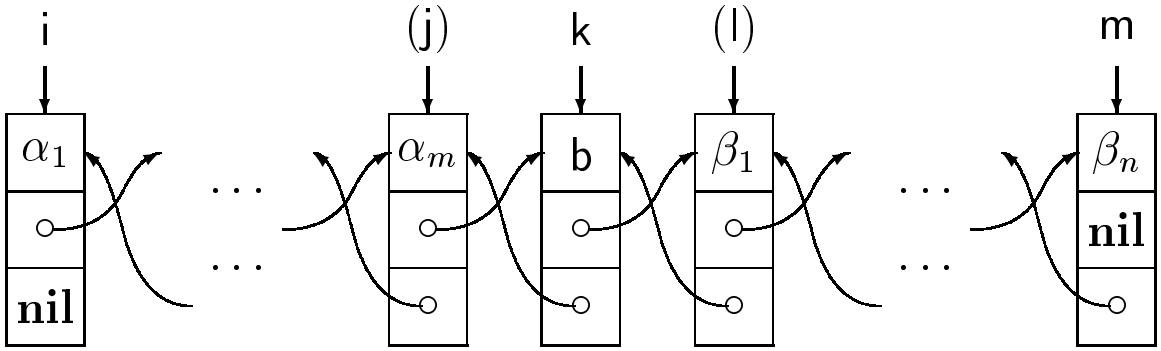
$\{\exists \mathsf{j}_0.\ \ \mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{j}_0, \mathsf{j})\ *\ \mathsf{dlist}\ \beta\ (\mathsf{j}_0, \mathsf{j}, \mathbf{nil}, \mathsf{m})\}$

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

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

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

Deleting an Element from a Doubly-Linked List



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

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

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

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

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

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

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

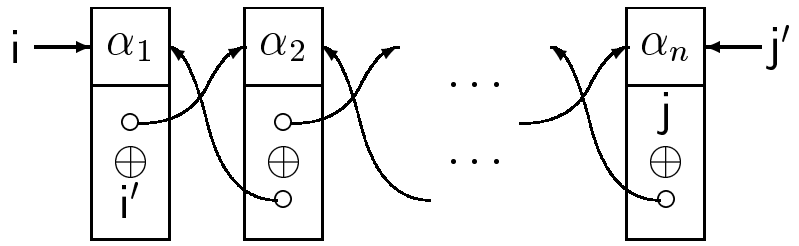
$\mathbf{setlpt}(\mathsf{l}, \mathsf{j}; \mathsf{m})$

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

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

An Inductive Definition: Xor-Linked Lists

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



is defined by

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

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

Properties

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

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

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

Emptyness Conditions (as with dlist)

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

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

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

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

A Set-Pointer Procedure for Xor-linked Lists

Since

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

the procedure

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

satisfies

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

Upon Reflection

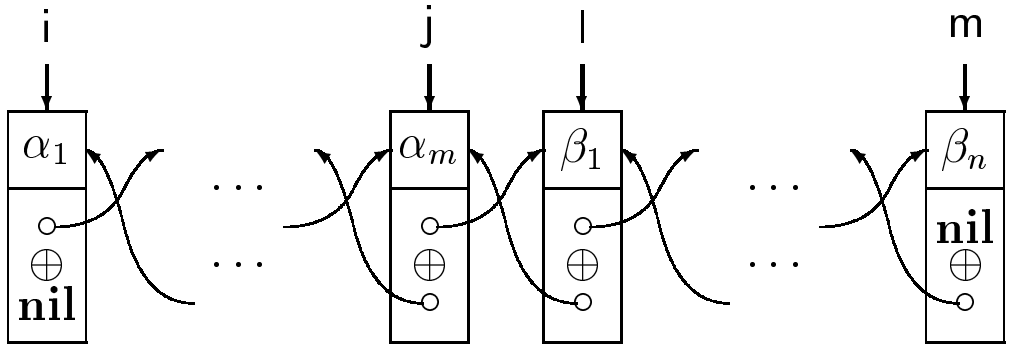
The procedure

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

satisfies

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

Inserting an Element into a Xor-Linked List



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

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

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

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

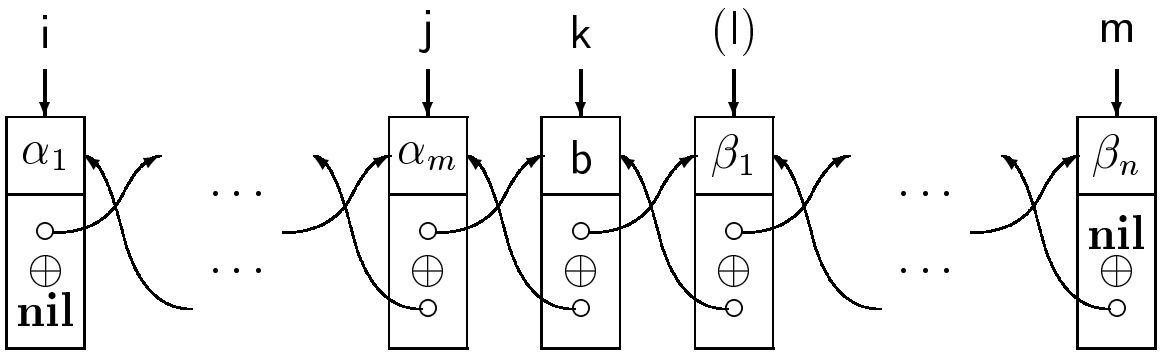
$\{\mathsf{xlist}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{k}, \mathsf{j})\ *\ \mathsf{k} \mapsto \mathsf{a}, (\mathsf{l} \oplus \mathsf{j})\ *\ \mathsf{xlist}\ \beta\ (\mathsf{l}, \mathsf{j}, \mathbf{nil}, \mathsf{m})\}$

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

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

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

Deleting an Element from a Xor-Linked List



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

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

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

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

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

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

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

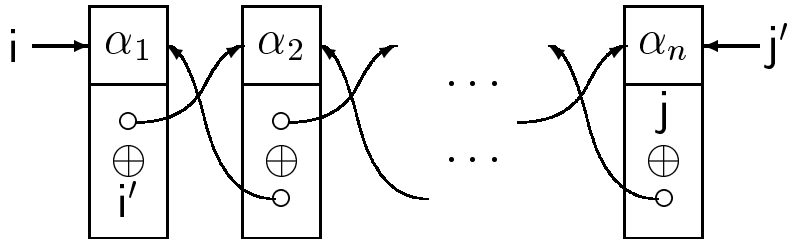
$\mathbf{xsetlpt}(\mathsf{l}, \mathsf{k}, \mathsf{j}; \mathsf{m})$

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

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

Reversing an Xor-Linked List

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



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

Proof (by induction on α):

Base case:

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

Induction step:

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

Specifications for Sorting by Merging

For sequences α and β, we define

$$\begin{array}{rcl} \beta \sim \alpha & \stackrel{\text{def}}{=} & \beta \text{ is a rearrangement of } \alpha, \text{ i.e., } \#\beta = \#\alpha \\ & & \text{and there is a permutation } f \text{ such that} \\ & & \forall k.\ 1 \le k \le \#\alpha \text{ implies } \beta_k = \alpha_{f(k)}. \\ \mathbf{ord}\ \beta & \stackrel{\text{def}}{=} & \beta \text{ is ordered in nonstrict increasing order.} \\ \mathsf{list}\ \alpha\ (e, -) & \stackrel{\text{def}}{=} & \exists x.\ \mathsf{list}\ \alpha\ (e, x). \end{array}$$

Then

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

and

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

A Proof for mergesort

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

A Proof for merge

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

Then

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

Thus we may define

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

A Proof for merge1

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