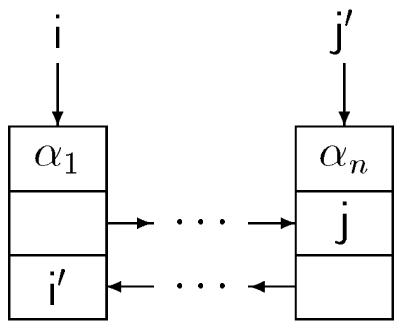
15–819A3 — Class Notes 2
Reasoning about Low-Level Programming Languages

John C. Reynolds
Department of Computer Science
Carnegie Mellon University

February 15, 2001 (revised)

An Inductive Definition: Doubly-Linked Lists

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



Definition

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

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

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

$$\mathsf{dlist}\ \alpha\,(\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') \Rightarrow \left(\begin{array}{rc} & (\mathsf{i} = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge \mathsf{j} = \mathbf{nil} \wedge \mathsf{i}' = \mathsf{j}')) \\ \wedge & (\mathsf{j}' = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge \mathsf{i}' = \mathbf{nil} \wedge \mathsf{i} = \mathsf{j})) \\ \wedge & (\mathsf{i} \neq \mathsf{j} \Rightarrow \alpha \neq \epsilon) \\ \wedge & (\mathsf{i}' \neq \mathsf{j}' \Rightarrow \alpha \neq \epsilon) \end{array}\right)$$

A Lookup-pointer Procedure for Doubly-linked Lists

Since

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

the procedure

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

satisfies

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

A Set-pointer Procedure for Doubly-linked Lists

Since

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

the procedure

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

satisfies

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

Upon Reflection

The procedure

$$\begin{array}{l}\mathsf{lookuplpt}(\mathsf{i},\mathsf{j}';\mathsf{i}') = \\ \quad \textbf{if } \mathsf{i} = \textbf{nil} \textbf{ then } \mathsf{i}' := \mathsf{j}' \textbf{ else } \mathsf{i}' := [\mathsf{i}+2]\end{array}$$

satisfies

$$\begin{array}{l}\{\textbf{dlist}\,\alpha\,(\mathsf{i},\mathsf{i}'_0,\textbf{nil},\mathsf{j}')\} \\ \mathsf{lookuplpt}(\mathsf{i},\mathsf{j}';\mathsf{i}') \\ \{\textbf{dlist}\,\alpha\,(\mathsf{i},\mathsf{i}'_0,\textbf{nil},\mathsf{j}') \wedge \mathsf{i}' = \mathsf{i}'_0\},\end{array}$$

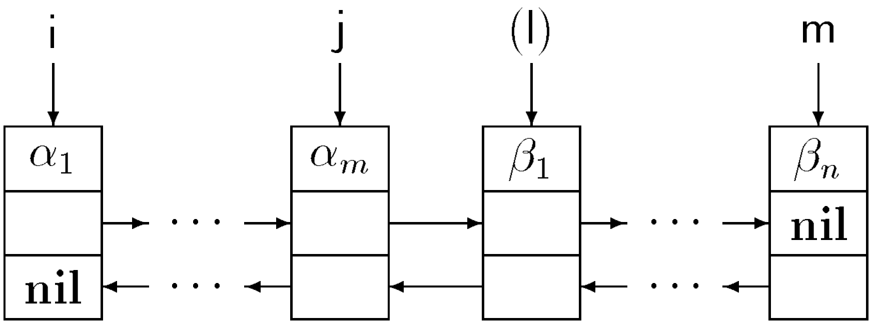
and the procedure

$$\begin{array}{l}\mathsf{setlpt}(\mathsf{i},\mathsf{i}';\mathsf{j}') = \\ \quad \textbf{if } \mathsf{i} = \textbf{nil} \textbf{ then } \mathsf{j}' := \mathsf{i}' \textbf{ else } [\mathsf{i}+2] := \mathsf{i}'\end{array}$$

satisfies

$$\begin{array}{l}\{\exists \mathsf{i}'.\ \ \textbf{dlist}\,\alpha\,(\mathsf{i},\mathsf{i}',\textbf{nil},\mathsf{j}')\} \\ \mathsf{setlpt}(\mathsf{i},\mathsf{i}';\mathsf{j}') \\ \{\textbf{dlist}\,\alpha\,(\mathsf{i},\mathsf{i}',\textbf{nil},\mathsf{j}')\}.\end{array}$$

Inserting an Element into a Doubly-Linked List



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

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

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

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

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

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

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

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

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

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

Deriving the Specification of the Call of $\mathsf{lookuprpt}$

From:

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

using Changing Free Variables:

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

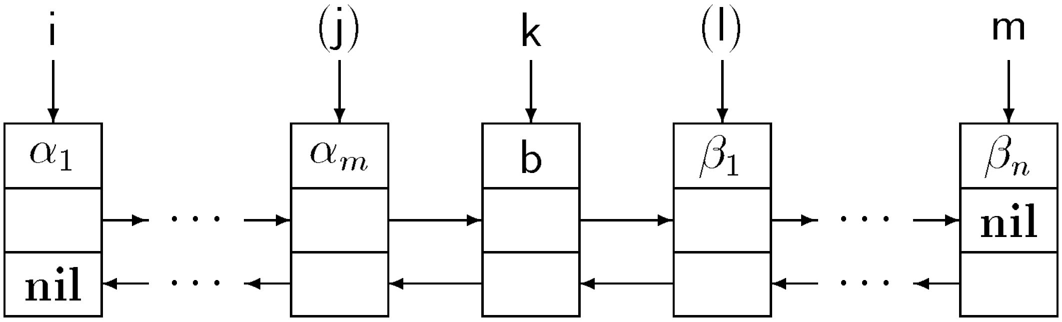
using Frame Introduction and an obvious property of equality:

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

using Existential Introduction, renaming, and the elimination of a vacuous quantifier:

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

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

Definition

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

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

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

$$\mathsf{xlist}\ \alpha\,(\mathsf{i},\mathsf{i}',\mathsf{j},\mathsf{j}') \Rightarrow \left(\begin{array}{ll} & (\mathsf{i}=\mathbf{nil} \Rightarrow (\alpha=\epsilon \wedge \mathsf{j}=\mathbf{nil} \wedge \mathsf{i}'=\mathsf{j}')) \\ \wedge & (\mathsf{j}'=\mathbf{nil} \Rightarrow (\alpha=\epsilon \wedge \mathsf{i}'=\mathbf{nil} \wedge \mathsf{i}=\mathsf{j})) \\ \wedge & (\mathsf{i} \neq \mathsf{j} \Rightarrow \alpha \neq \epsilon) \\ \wedge & (\mathsf{i}' \neq \mathsf{j}' \Rightarrow \alpha \neq \epsilon) \end{array}\right)$$

A Set-pointer Procedure for Xor-linked Lists

Since

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

the procedure

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

satisfies

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

Upon Reflection

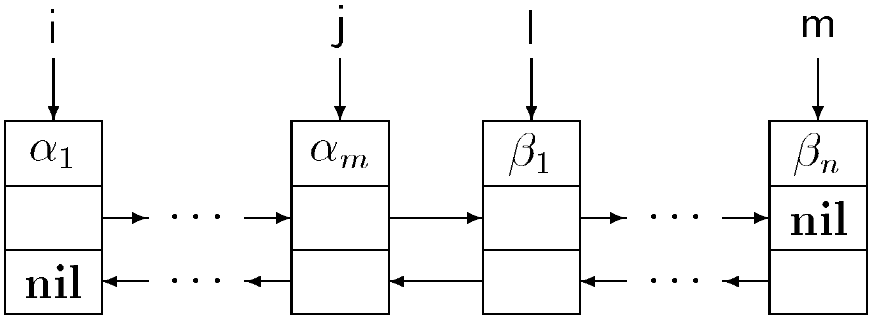
The procedure

$$\begin{aligned}&\mathsf{xsetlpt}(\mathsf{i},\mathsf{i}',\mathsf{k};\mathsf{j}') = \\ &\quad \textbf{if } \mathsf{i} = \textbf{nil then } \mathsf{j}' := \mathsf{k} \textbf{ else} \\ &\qquad \textbf{newvar } \mathsf{x} \textbf{ in } (\mathsf{x} := [\mathsf{i}+1]\,;\,[\mathsf{i}+1] := \mathsf{x} \oplus \mathsf{i}' \oplus \mathsf{k})\end{aligned}$$

satisfies

$$\begin{aligned}&\{\textbf{xlist}\,\alpha\,(\mathsf{i},\mathsf{i}',\textbf{nil},\mathsf{j}')\} \\ &\mathsf{xsetlpt}(\mathsf{i},\mathsf{i}',\mathsf{k};\mathsf{j}') \\ &\{\textbf{xlist}\,\alpha\,(\mathsf{i},\mathsf{k},\textbf{nil},\mathsf{j}')\}.\end{aligned}$$

Inserting an Element into a Xor-Linked List



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

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

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

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

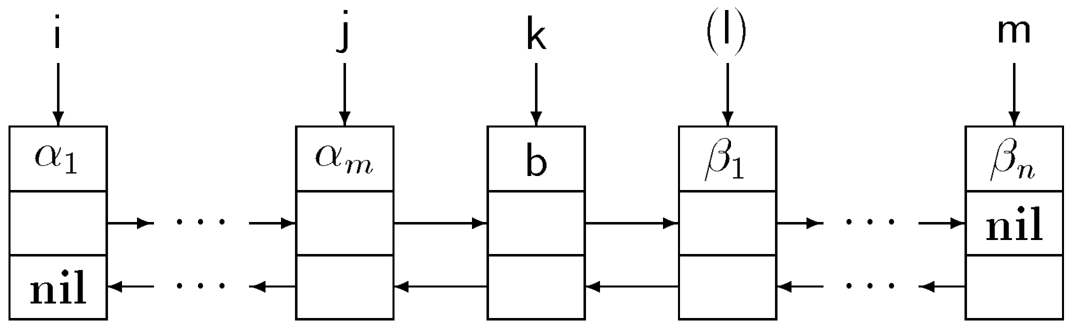
$$\{\mathbf{xlist}\ \alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{k}, \mathsf{j})\ *\ \mathsf{k} \mapsto \mathsf{a}, (\mathsf{l} \oplus \mathsf{j})\ *\ \mathbf{xlist}\ \beta\,(\mathsf{l}, \mathsf{j}, \mathbf{nil}, \mathsf{m})\}$$

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

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

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

Deleting an Element from a Xor-Linked List



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

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

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

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

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

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

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

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

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

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