Further Reasoning about Shared Mutable Data Structure

John C. Reynolds

Department of Computer Science
Carnegie Mellon University
and
BRICS, Department of Computer Science
University of Aarhus

Joint work with
Peter W. O'Hearn and Hongseok Yang

June 6, 2001 (corrected June 11)

Basic Ideas

- Extension of Hoare logic for imperative programs
- Low-level programming language
  - Store/heap dichotomy (both updatable)
  - Heap-manipulation commands (not expressions)
  - Heap faults
  - Nondeterministic allocation
- Independent (spatial) conjunction ($*$)
- Inductive definitions over abstract structures
- Recent extensions
  - Unrestricted address arithmetic
  - Iterated independent conjunction (arrays)
  - Assertion variables

References

[1] Rodney M. Burstall. Some techniques for proving correctness of programs which alter data structures. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 7*, pages 23–50. Edinburgh University Press, Edinburgh, Scotland, 1972.

[2] Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 1999.

[3] John C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In Jim Davies, Bill Roscoe, and Jim Woodcock, editors, *Millennial Perspectives in Computer Science*, pages 303–321, Houndsmill, Hampshire, 2000. Palgrave.

[4] Samin Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 14–26, New York, 2001. ACM.

[5] John C. Reynolds and Peter W. O'Hearn. Reasoning about shared mutable data structure (abstract of invited lecture). In Fritz Henglein, John Hughes, Henning Makholm, and Henning Niss, editors, *SPACE 2001: Informal Proceedings of Workshop on Semantics, Program Analysis and Computing Environments for Memory Management*, page 7. IT University of Copenhagen, 2001. The slides for this lecture are available at ftp://ftp.cs.cmu.edu/user/jcr/spacetalk.ps.gz.

New Commands

$$
\begin{aligned}
\langle\text{comm}\rangle ::= & \cdots \\
& \mid \langle\text{var}\rangle := \mathbf{cons}(\langle\text{exp}\rangle, \ldots, \langle\text{exp}\rangle) \\
& \mid \langle\text{var}\rangle := [\langle\text{exp}\rangle] \\
& \mid [\langle\text{exp}\rangle] := \langle\text{exp}\rangle \\
& \mid \mathbf{dispose}\, \langle\text{exp}\rangle
\end{aligned}
$$

New Assertions

$$
\begin{aligned}
\langle\text{assert}\rangle ::= & \cdots \\
& \mid \langle\text{exp}\rangle \mapsto \langle\text{exp}\rangle \\
& \mid \mathbf{emptyheap} \\
& \mid \langle\text{assert}\rangle \;*\; \langle\text{assert}\rangle \\
& \mid \langle\text{assert}\rangle \mathrel{-\!\!*} \langle\text{assert}\rangle
\end{aligned}
$$

Why **cons**, **car**, and **cdr** do not occur in expressions

$$(\mathsf{x} = \mathsf{x})/\mathsf{x} \to \mathbf{cons}(1,2) \;\text{ is }\; \mathbf{cons}(1,2) = \mathbf{cons}(1,2)$$

$$(\mathsf{x} = \mathsf{x} \;*\; \mathsf{y} = \mathsf{y})/\mathsf{x} \to \mathbf{car}(\mathsf{z}), \mathsf{y} \to \mathbf{car}(\mathsf{z}) \;\text{ is}$$

$$(\mathbf{car}(\mathsf{z}) = \mathbf{car}(\mathsf{z})) \;*\; (\mathbf{car}(\mathsf{z}) = \mathbf{car}(\mathsf{z}))$$

Stores and Heaps (Old)

$$
\begin{aligned}
\text{Values} &= \mathbf{Z} + \text{Atoms} + \text{Locations} \\
\textbf{nil} &\in \text{Atoms} \\
\text{States} &= \text{Stores} \times \text{Heaps} \\
\text{Stores} &= \langle\text{var}\rangle \to \text{Values} \\
\text{Heaps} &= \bigcup_{A \overset{\text{fin}}{\subseteq} \text{Locations}} (A \to \text{Values}^{+})
\end{aligned}
$$

Stores and Heaps (New)

$$
\begin{aligned}
\text{Atoms} &\subseteq \mathbf{Z} \\
\text{Locations} &\subseteq \mathbf{Z} \\
\text{Atoms} \cap \text{Locations} &= \{\} \\
\textbf{nil} &\in \text{Atoms} \\
\text{States} &= \text{Stores} \times \text{Heaps} \\
\text{Stores} &= \langle\text{var}\rangle \to \mathbf{Z} \\
\text{Heaps} &= \bigcup_{A \overset{\text{fin}}{\subseteq} \text{Locations}} (A \to \mathbf{Z})
\end{aligned}
$$

Semantic Equations

$$
\begin{aligned}
[\![-]\!]_{\text{exp}} &\in \langle\text{exp}\rangle \to \text{Stores} \to \mathbf{Z} \\
[\![-]\!]_{\text{boolexp}} &\in \langle\text{boolexp}\rangle \to \text{Stores} \to \mathbf{B} \\
[\![-]\!]_{\text{assert}} &\in \langle\text{assert}\rangle \to \text{States} \to \mathbf{B}
\end{aligned}
$$

Structural Operational Semantics

An execution (of a command) is a sequence

$$\gamma_0 \rightsquigarrow \gamma_1 \rightsquigarrow \cdots$$

of *configurations* γ_i related by the *transition relation* $\rightsquigarrow$.

A configuration is either:

- *nonterminal*: A command-state pair.
- *terminal*: A state or **abort**.

For example:

$$\begin{gathered}
\langle \mathsf{x} := \mathbf{cons}(1,2)\ ;\ \mathsf{y} := [\mathsf{x}]\ ;\ [\mathsf{x}+1] := 3, ([\,\mathsf{x}\colon 3 \mid \mathsf{y}\colon 4\,], [\,])\rangle \rightsquigarrow \\
\langle \mathsf{y} := [\mathsf{x}]\ ;\ [\mathsf{x}+1] := 3, ([\,\mathsf{x}\colon 37 \mid \mathsf{y}\colon 4\,], [\,37\colon 1 \mid 38\colon 2\,])\rangle \rightsquigarrow \\
\langle [\mathsf{x}+1] := 3, ([\,\mathsf{x}\colon 37 \mid \mathsf{y}\colon 1\,], [\,37\colon 1 \mid 38\colon 2\,])\rangle \rightsquigarrow \\
([\,\mathsf{x}\colon 37 \mid \mathsf{y}\colon 1\,], [\,37\colon 1 \mid 38\colon 3\,]).
\end{gathered}$$

$$\begin{gathered}
\langle \mathsf{x} := \mathbf{cons}(1,2)\ ;\ \mathsf{y} := [\mathsf{x}+2]\ ;\ [\mathsf{x}+1] := 3, ([\,\mathsf{x}\colon 3 \mid \mathsf{y}\colon 4\,], [\,])\rangle \rightsquigarrow \\
\langle \mathsf{y} := [\mathsf{x}+2]\ ;\ [\mathsf{x}+1] := 3, ([\,\mathsf{x}\colon 37 \mid \mathsf{y}\colon 4\,], [\,37\colon 1 \mid 38\colon 2\,])\rangle \rightsquigarrow \\
\mathbf{abort}.
\end{gathered}$$

Our semantics is total, but nondeterminate.

Transition Rules for the New Commands

- Construction

$$\langle v := \mathbf{cons}(e_0, \ldots, e_{n-1}), (s, h)\rangle$$
$$\rightsquigarrow ([\, s \mid v{:}\, \ell \,], [\, h \mid \ell{:}\, [\![e_0]\!]_{\mathrm{exp}} s \mid \ldots \mid \ell + n - 1{:}\, [\![e_{n-1}]\!]_{\mathrm{exp}} s \,])$$
$$\text{where } \ell, \ldots, \ell + n - 1 \in \mathrm{Locations} - \mathrm{dom}\, h$$

- Lookup

$$\langle v := [e], (s, h)\rangle \rightsquigarrow ([\, s \mid v{:}\, h([\![e]\!]_{\mathrm{exp}} s) \,], h)$$
$$\text{when } [\![e]\!]_{\mathrm{exp}} s \in \mathrm{dom}\, h$$

$$\langle v := [e], (s, h)\rangle \rightsquigarrow \mathbf{abort} \qquad \text{when } [\![e]\!]_{\mathrm{exp}} s \notin \mathrm{dom}\, h$$

- Mutation

$$\langle [e] := e', (s, h)\rangle \rightsquigarrow (s, [\, h \mid [\![e]\!]_{\mathrm{exp}} s{:}\, [\![e']\!]_{\mathrm{exp}} s \,])$$
$$\text{when } [\![e]\!]_{\mathrm{exp}} s \in \mathrm{dom}\, h$$

$$\langle [e] := e', (s, h)\rangle \rightsquigarrow \mathbf{abort} \qquad \text{when } [\![e]\!]_{\mathrm{exp}} s \notin \mathrm{dom}\, h$$

- Disposal

$$\langle \mathbf{dispose}\ e, (s, h)\rangle \rightsquigarrow (s, h \rceil (\mathrm{dom}\, h - \{[\![e]\!]_{\mathrm{exp}} s\}))$$
$$\text{when } [\![e]\!]_{\mathrm{exp}} s \in \mathrm{dom}\, h$$

$$\langle \mathbf{dispose}\ e, (s, h)\rangle \rightsquigarrow \mathbf{abort} \qquad \text{when } [\![e]\!]_{\mathrm{exp}} s \notin \mathrm{dom}\, h$$

Semantics of the New Assertions

$$[\![-]\!]_{\text{assert}} \in \langle \text{assert} \rangle \to (\text{Stores} \times \text{Heaps}) \to \mathbf{B}.$$

$[\![e \mapsto e']\!]_{\text{assert}}(s, h)$ iff

$$\operatorname{dom} h = \{[\![e]\!]_{\exp} s\} \text{ and } h([\![e]\!]_{\exp} s) = [\![e']\!]_{\exp} s,$$

$[\![\mathbf{emptyheap}]\!]_{\text{assert}}(s, h)$ iff

$$\operatorname{dom} h = \{\},$$

$[\![p_0 \ * \ p_1]\!]_{\text{assert}}(s, h)$ iff

$$\exists h_0, h_1.\ h_0 \perp h_1 \text{ and } h_0 \cdot h_1 = h \text{ iff}$$
$$\text{and } [\![p_0]\!]_{\text{assert}}(s, h_0) \text{ and } [\![p_1]\!]_{\text{assert}}(s, h_1),$$

$[\![p_0 \mathrel{-\!\!*} p_1]\!]_{\text{assert}}(s, h)$ iff

$$\forall h'.\ (h' \perp h \text{ and } [\![p_0]\!]_{\text{assert}}(s, h'))$$
$$\text{implies } [\![p_1]\!]_{\text{assert}}(s, h \cdot h'),$$

where

$$h_0 \perp h_1 \ \text{ iff } \ \operatorname{dom} h_0 \cap \operatorname{dom} h_1 = \{\}$$

$$\operatorname{dom}(h_0 \cdot h_1) \ = \ \operatorname{dom} h_0 \cup \operatorname{dom} h_1$$

$$(h_0 \cdot h_1)\ell \ = \ \begin{cases} h_0 \ell & \text{when } \ell \in \operatorname{dom} h_0 \\ h_1 \ell & \text{when } \ell \in \operatorname{dom} h_1. \end{cases}$$

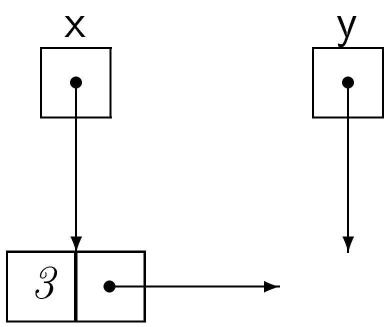
Some Abbreviations

$$
\begin{aligned}
e \mapsto - &\stackrel{\text{def}}{=} \exists x'.\ e \mapsto x' \quad \text{where } x' \text{ not free in } e \\
e \hookrightarrow e' &\stackrel{\text{def}}{=} e \mapsto e' \ast \mathbf{true} \\
e \mapsto e_0, \ldots, e_{n-1} &\stackrel{\text{def}}{=} e \mapsto e_0 \ast \cdots \ast e + n - 1 \mapsto e_{n-1} \\
e \hookrightarrow e_0, \ldots, e_{n-1} &\stackrel{\text{def}}{=} e \hookrightarrow e_0 \ast \cdots \ast e + n - 1 \hookrightarrow e_{n-1} \\
&\quad \text{iff } e \mapsto e_0, \ldots, e_{n-1} \ast \mathbf{true}
\end{aligned}
$$

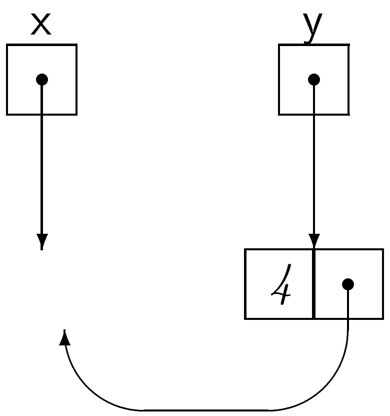
An Example of $*$

$$[\![\mathsf{x} \mapsto 3, \mathsf{y} \ * \ \mathsf{y} \mapsto 4, \mathsf{x}]\!]_{\text{assert}}(s, h)$$
$$\text{iff } \exists h_0, h_1.\ h_0 \perp h_1 \text{ and } h_0 \cdot h_1 = h$$
$$\text{and } [\![\mathsf{x} \mapsto 3, \mathsf{y}]\!]_{\text{assert}}(s, h_0) \text{ and } [\![\mathsf{y} \mapsto 4, \mathsf{x}]\!]_{\text{assert}}(s, h_1)$$

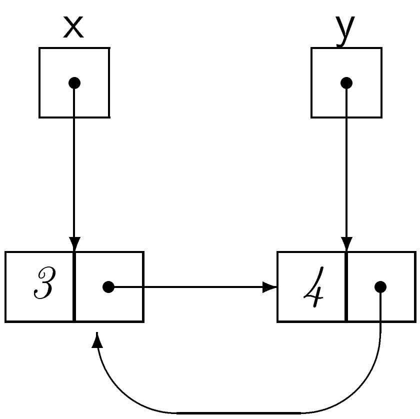
$\mathsf{x} \mapsto 3, \mathsf{y}$



$\mathsf{y} \mapsto 4, \mathsf{x}$



$\mathsf{x} \mapsto 3, \mathsf{y} \ * \ \mathsf{y} \mapsto 4, \mathsf{x}$

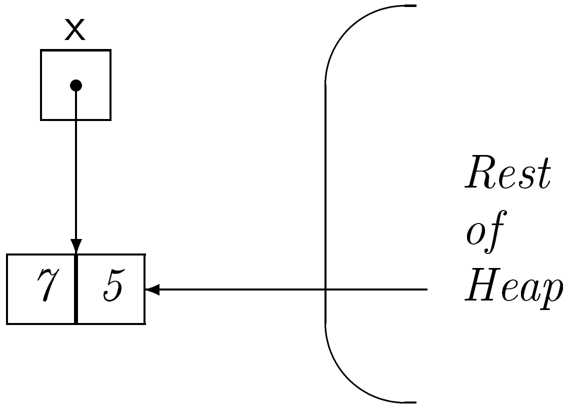


An Example of $-\!\!*$

$$\begin{aligned}&[\![\mathsf{x} \mapsto 7,5 -\!\!* P]\!]_{\mathrm{assert}}(s,h)\\ &\qquad \text{iff } \forall h'.\ (h' \perp h \text{ and } [\![\mathsf{x} \mapsto 7,5]\!]_{\mathrm{assert}}(s,h'))\\ &\qquad\qquad\qquad\qquad \text{implies } [\![P]\!]_{\mathrm{assert}}(s,h\cdot h')\end{aligned}$$

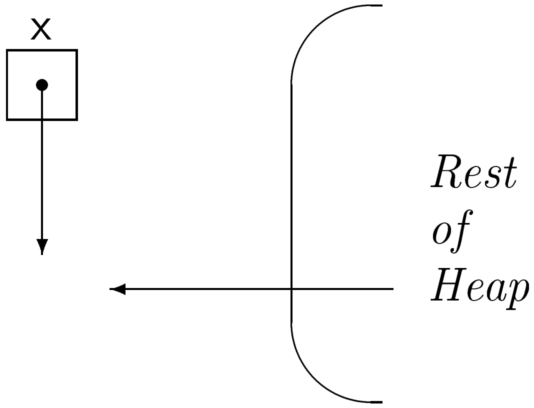
If:

P



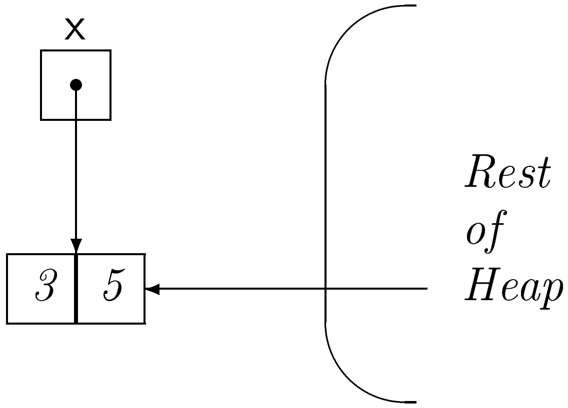
then:

$\mathsf{x} \mapsto 7,5 -\!\!* P$



and:

$\mathsf{x} \mapsto 3,5 \ * \ (\mathsf{x} \mapsto 7,5 -\!\!* P)$



Rules and Axiom Schemata for $*$ and $-\!\!*$

$$p_1 * p_2 \Leftrightarrow p_2 * p_1$$
$$(p_1 * p_2) * p_3 \Leftrightarrow p_1 * (p_2 * p_3)$$
$$p * \textbf{emptyheap} \Leftrightarrow p$$
$$(p_1 \vee p_2) * q \Leftrightarrow (p_1 * q) \vee (p_2 * q)$$
$$(p_1 \wedge p_2) * q \Rightarrow (p_1 * q) \wedge (p_2 * q)$$
$$(\exists x.\ p_1) * p_2 \Leftrightarrow \exists x.\ (p_1 * p_2) \quad \text{when } x \text{ not free in } p_2$$
$$(\forall x.\ p_1) * p_2 \Rightarrow \forall x.\ (p_1 * p_2) \quad \text{when } x \text{ not free in } p_2$$

$$\frac{p_1 \Rightarrow p_2 \qquad q_1 \Rightarrow q_2}{p_1 * q_1 \Rightarrow p_2 * q_2}$$

$$\frac{p_1 * p_2 \Rightarrow p_3}{p_1 \Rightarrow (p_2 -\!\!* p_3)} \qquad\qquad \frac{p_1 \Rightarrow (p_2 -\!\!* p_3)}{p_1 * p_2 \Rightarrow p_3}$$

Axiom Schemata for $\mapsto$

$$x \mapsto e \wedge x' \mapsto e' \Leftrightarrow x \mapsto e \wedge x = x' \wedge e = e'$$
$$x \mapsto e * x' \mapsto e' * \textbf{true} \Rightarrow x \neq x'$$
$$\textbf{emptyheap} \Leftrightarrow \forall x.\ \neg(x \mapsto - * \textbf{true}).$$

Pure Assertions

An assertion is *pure* if it does not contain $\mapsto$ or **emptyheap**.

A pure assertion does not depend upon the heap.

Axiom Schemata for Purity

$$
\begin{aligned}
p_1 \wedge p_2 \Rightarrow p_1 \ast p_2 &\quad \text{when } p_1 \text{ or } p_2 \text{ is pure} \\
p_1 \ast p_2 \Rightarrow p_1 \wedge p_2 &\quad \text{when } p_1 \text{ and } p_2 \text{ are pure} \\
(p \wedge q) \ast r \Leftrightarrow (p \ast r) \wedge q &\quad \text{when } q \text{ is pure} \\
(p_1 -\!\!\ast\ p_2) \Rightarrow (p_1 \Rightarrow p_2) &\quad \text{when } p_1 \text{ is pure} \\
(p_1 \Rightarrow p_2) \Rightarrow (p_1 -\!\!\ast\ p_2) &\quad \text{when } p_1 \text{ and } p_2 \text{ are pure}
\end{aligned}
$$

In addition, the model validates all of the laws of classical logic for $\neg$, $\Rightarrow$, $\wedge$ and $\vee$. It uses the boolean algebra structure of the powerset of Heaps, which amounts to pointwise interpretations for each classical connective.

Two Unsound Axiom Schemata

$$p \not\Rightarrow p * p \qquad \text{(Contraction)}$$

e.g. $p : \mathsf{x} \mapsto 1$

$$p * q \not\Rightarrow p \qquad \text{(Weakening)}$$

e.g. $p : \mathsf{x} \mapsto 1$

$q : \mathsf{y} \mapsto 2$

Specifications

$$[\![-]\!]_{\text{spec}} \in \langle\text{spec}\rangle \to \mathbf{B}.$$

Partial correctness:

$$\begin{aligned}
&[\![\{p\}\ c\ \{q\}]\!]_{\text{spec}} \text{ iff } \forall\sigma \in \text{States}.\ [\![p]\!]_{\text{assert}}\sigma \text{ implies}\\
&\qquad \neg\,(c, \sigma \leadsto^* \mathbf{abort})\\
&\qquad \text{and } (\forall\sigma'.\ c, \sigma \leadsto^* \sigma' \text{ implies } [\![q]\!]_{\text{assert}}\sigma')
\end{aligned}$$

Total correctness:

$$\begin{aligned}
&[\![[\ p\ ]\ c\ [\ q\ ]]\!]_{\text{spec}} \text{ iff } \forall\sigma \in \text{States}.\ [\![p]\!]_{\text{assert}}\sigma \text{ implies}\\
&\qquad \neg\,(c, \sigma \leadsto^* \mathbf{abort})\\
&\qquad \text{and } \neg\,(c, \sigma \uparrow)\\
&\qquad \text{and } (\forall\sigma'.\ c, \sigma \leadsto^* \sigma' \text{ implies } [\![q]\!]_{\text{assert}}\sigma').
\end{aligned}$$

- Specifications are implicitly quantified over both stores and heaps — and over all executions.
- An execution giving a memory fault falsifies a specification.

Examples:

$$\{\mathbf{emptyheap}\}\ \mathsf{x} := \mathbf{cons}(1, 2)\ \{\mathsf{x} \mapsto 1, 2\}$$

$$\{\mathsf{x} \mapsto 1, 2\}\ \mathsf{y} := [\mathsf{x}]\ ;\ [\mathsf{x} + 1] := 3\ \{\mathsf{x} \mapsto 1, 3 \wedge \mathsf{y} = 1\}$$

Violating Record Boundaries

Suppose P holds for a state (s, h), and C' does not assign to x. Then

$$[\,P\,]\ C\ ;\mathsf{x} := \mathbf{cons}(1,2)\ ;C'\ ;[\mathsf{x}+2] := 3\ ;C''\ [\,Q\,]$$

cannot hold. For suppose the contrary. Then

$$\langle C, (s,h)\rangle \rightsquigarrow^* (s', h').$$

Let ℓ be such that $\ell, \ell+1, \ell+2 \notin \mathrm{dom}\, h'$. Then

$$\langle \mathsf{x} := \mathbf{cons}(1,2), (s', h')\rangle \rightsquigarrow ([\,s' \mid \mathsf{x}\colon \ell\,], [\,h' \mid \ell\colon 1 \mid \ell+1\colon 2\,]),$$

and

$$\langle C', ([\,s' \mid \mathsf{x}\colon \ell\,], [\,h' \mid \ell\colon 1 \mid \ell+1\colon 2\,])\rangle \rightsquigarrow^* (s'', h''),$$

where $s''\mathsf{x} = \ell$ and $\ell + 2 \notin \mathrm{dom}\, h''$. Then

$$\langle [\mathsf{x}+2] := 3, (s'', h'')\rangle \rightsquigarrow \mathbf{abort},$$

and

$$\langle [\mathsf{x}+2] := 3\ ; C'', (s'', h'')\rangle \rightsquigarrow \mathbf{abort},$$

contradicting the assumption.

An Inference Rule for Mutation

$$\{(e \mapsto -) \ast p\}$$
$$[e] := e'$$
$$\{(e \mapsto e') \ast p\}$$

An Inference Rule for Construction

$$\{p\}$$
$$v := \mathbf{cons}(e_0, \ldots, e_{n-1})$$
$$\{(v \mapsto e_0, \ldots, e_{n-1}) \ast p\}$$

where v is not free in e_0, $\ldots$, e_{n-1}, or p.

A Stronger Rule

$$\{p\}$$
$$v := \mathbf{cons}(e_0, \ldots, e_{n-1})$$
$$\{\exists v'.\ (v \mapsto e'_0, \ldots, e'_{n-1}) \ast p'\}$$

where

- v' is distinct from v and not free in e_0, $\ldots$, e_{n-1}, or p,
- p' is the substitution $p/v \to v'$,
- each e'_i is the substitution $e_i/v \to v'$.

An Inference Rule for Lookup

$$\begin{array}{l} \{\exists v'.\ e \hookrightarrow v' \wedge p'\} \\ v := [e] \\ \{p\} \end{array}$$

where v' does not occur free in p or e, and p' is $p/v \to v'$.

Disposal

$$\begin{array}{l} \{(e \mapsto -) \ast p\} \\ \textbf{dispose}\ e \\ \{p\} \end{array}$$

Frame Rule

$$\frac{\{p\}\ c\ \{q\}}{\{p \ast r\}\ c\ \{q \ast r\}}$$

when no variable occurs both free in r and assigned by c.

Free Variable Substitution

$$\frac{\{p\}\ c\ \{q\}}{(\{p\}\ c\ \{q\})/v_1 \to e_1,\ \ldots\ ,\ v_n \to e_n},$$

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

For example, in

$$\{\mathsf{x} = \mathsf{y}\}\ \mathsf{x} := \mathsf{x} + \mathsf{y}\ \{\mathsf{x} = 2 \times \mathsf{y}\},$$

one can substitute $\mathsf{x}, \mathsf{y}\ \to\ \mathsf{z}, 2 \times \mathsf{w} - 1$ to infer

$$\{\mathsf{z} = 2 \times \mathsf{w} - 1\}\ \mathsf{z} := \mathsf{z} + 2 \times \mathsf{w} - 1\ \{\mathsf{z} = 2 \times (2 \times \mathsf{w} - 1)\}.$$

But one cannot substitute $\mathsf{x}, \mathsf{y}\ \to\ \mathsf{z}, 2 \times \mathsf{z} - 1$ to infer

$$\{\mathsf{z} = 2 \times \mathsf{z} - 1\}\ \mathsf{z} := \mathsf{z} + 2 \times \mathsf{z} - 1\ \{\mathsf{z} = 2 \times (2 \times \mathsf{z} - 1)\}.$$

Auxiliary Variable Elimination

$$\frac{\{p\}\ c\ \{q\}}{\{\exists v.\ p\}\ c\ \{\exists v.\ q\}},$$

where v is not free in c.

For all of the new inference rules, there are similar rules for total correctness.

Simple Procedures

A 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. (The free variables assigned by the call are $y_1, \cdots, y_n$.)

Recursion

For partial correctness, if c contains recursive calls, one can also use the assumption

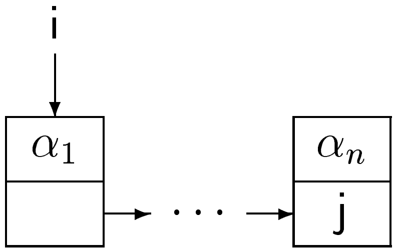
$$\{p\}\ h(x_1, \cdots, x_m; y_1, \cdots, y_n)\ \{q\}$$

in proving

$$\{p\}\ c\ \{q\}.$$

Inductive Definitions: Singly-Linked Lists

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

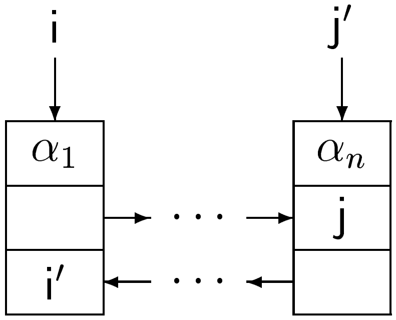


$$\mathsf{list}\ \epsilon\ (\mathsf{i}, \mathsf{j}) \stackrel{\mathrm{def}}{=} \mathbf{emptyheap} \land \mathsf{i} = \mathsf{j}$$

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

Doubly-Linked Lists

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



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

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

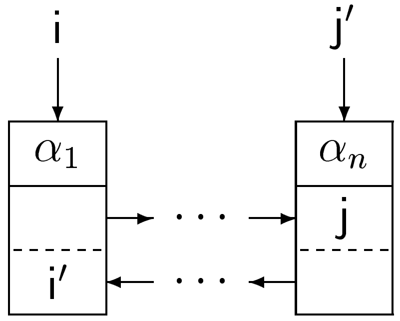
Xor-Linked Lists

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

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

Xor-Linked Lists

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



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

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

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

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

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

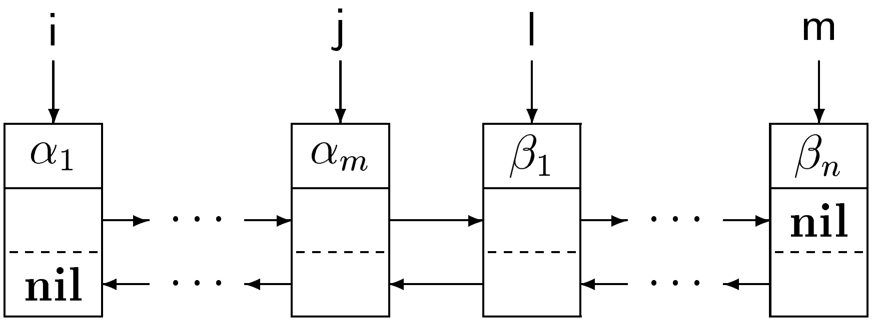
The procedure

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

satisfies

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

Inserting an Element into a Xor-Linked List



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

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

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

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

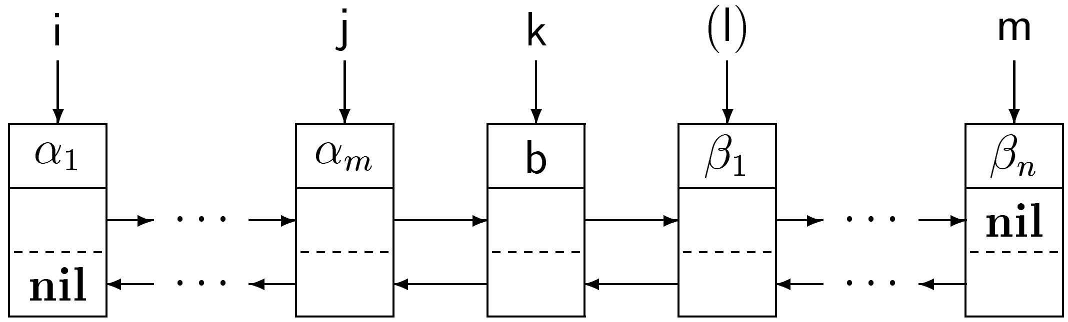
$\{\textbf{xlist}\,\alpha\,(\mathsf{i},\textbf{nil},\mathsf{k},\mathsf{j})\ *\ \mathsf{k}\mapsto\mathsf{a},(\mathsf{l}\oplus\mathsf{j})\ *\ \textbf{xlist}\,\beta\,(\mathsf{l},\mathsf{j},\textbf{nil},\mathsf{m})\}$

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

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

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

Deleting an Element from a Xor-Linked List



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

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

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

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

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

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

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

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

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

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

Iterated Independent Conjunction

$$\langle\text{assert}\rangle ::= \cdots \mid \bigodot_{\langle\text{var}\rangle=\langle\text{exp}\rangle}^{\langle\text{exp}\rangle} \langle\text{assert}\rangle$$

$$\begin{aligned}
&[\![\textstyle\bigodot_{x=e}^{e'} p]\!]_{\text{assert}}(s,h) = \\
&\qquad \textbf{let } m = [\![e]\!]_{\text{exp}} s,\ n = [\![e']\!]_{\text{exp}} s,\ I = \{\, i \mid m \leq i \leq n \,\} \textbf{ in} \\
&\qquad\qquad \exists H \in I \rightarrow \text{Heaps}. \\
&\qquad\qquad\qquad \forall i, j \in I.\ i \neq j \text{ implies } Hi \perp Hj \\
&\qquad\qquad\qquad \text{and } h = \bigcup\{\, Hi \mid i \in I \,\} \\
&\qquad\qquad\qquad \text{and } \forall i \in I.\ [\![p]\!]_{\text{assert}}([\, s \mid v{:}\, i \,], Hi).
\end{aligned}$$

Axiom Schemata

$$m > n \Rightarrow \Big(\textstyle\bigodot_{i=m}^{n} p(i) \Leftrightarrow \textbf{emptyheap}\Big)$$

$$m = n \Rightarrow \Big(\textstyle\bigodot_{i=m}^{n} p(i) \Leftrightarrow p(m)\Big)$$

$$k \leq m \leq n+1 \Rightarrow \Big(\textstyle\bigodot_{i=k}^{n} p(i) \Leftrightarrow \Big(\textstyle\bigodot_{i=k}^{m-1} p(i) \ * \ \textstyle\bigodot_{i=m}^{n} p(i)\Big)\Big)$$

$$\textstyle\bigodot_{i=m}^{n} p(i) \Leftrightarrow \textstyle\bigodot_{i=m-k}^{n-k} p(i+k)$$

$$m \leq n \Rightarrow \Big(\Big(\textstyle\bigodot_{i=m}^{n} p(i)\Big) \ * \ q \Leftrightarrow \textstyle\bigodot_{i=m}^{n}(p(i) \ * \ q)\Big)$$

when q is pure

$$m \leq j \leq n \Rightarrow \Big(\Big(\textstyle\bigodot_{i=m}^{n} p(i)\Big) \Rightarrow (p(j) \ * \ \textbf{true})\Big)$$

Array Allocation

$$\langle\text{comm}\rangle ::= \cdots \mid \langle\text{var}\rangle := \textbf{allocate}\ \langle\text{exp}\rangle$$

$$\langle v := \textbf{allocate}\ e, (s, h)\rangle \rightsquigarrow ([\, s \mid v{:}\, \ell\,], h \cdot h')$$
$$\text{where } h \perp h' \text{ and } \operatorname{dom} h' = \{\, i \mid \ell \le i \le \ell + [\![e]\!]_{\text{exp}} s \,\}.$$

An Inference Rule

$$\{p\}$$
$$v := \textbf{allocate}\ e$$
$$\{p \;*\; \textstyle\bigodot_{i=v}^{v+e-1} i \mapsto -\},$$

where v does not occur free in p or e.

Specifying Subset Lists

Let $\alpha, \beta \in \mathbf{Z}^*$ and $\sigma \in (\mathbf{Z}^*)^*$ in

$\{\mathsf{list}\ \alpha\ (\mathsf{i}, \mathbf{nil})\}$
"Set j to list of lists of subsets of i"
$\{\exists \sigma, \beta.\ \ \mathsf{ss}(\alpha^\dagger, \sigma) \wedge (\mathsf{list}\ \beta\ (\mathsf{j}, \mathbf{nil})\ * \ (Q(\sigma, \beta) \wedge R(\beta)))\}$,

where

$$\begin{aligned}
\#\mathsf{ext_a}\sigma &\stackrel{\text{def}}{=} \#\sigma \\
(\mathsf{ext_a}\sigma)_i &\stackrel{\text{def}}{=} \mathsf{a}{\cdot}\sigma_i
\end{aligned}$$

$$\begin{aligned}
\mathsf{ss}(\epsilon, \sigma) &\stackrel{\text{def}}{=} \sigma = [\epsilon] \\
\mathsf{ss}(\mathsf{a}{\cdot}\alpha, \sigma) &\stackrel{\text{def}}{=} \exists \sigma'.\ \ \mathsf{ss}(\alpha, \sigma') \wedge \sigma = (\mathsf{ext_a}\sigma')^\dagger{\cdot}\sigma'
\end{aligned}$$

$$Q(\sigma, \beta) \stackrel{\text{def}}{=} \#\beta = \#\sigma \wedge \forall_{\mathsf{i}=1}^{\#\beta}(\mathsf{list}\ \sigma_\mathsf{i}\ (\beta_\mathsf{i}, \mathbf{nil})\ *\ \mathbf{true})$$

$$\begin{aligned}
R(\beta) \stackrel{\text{def}}{=}\ & (\beta_{\#\beta} = \mathbf{nil} \wedge \mathbf{emptyheap})\ * \\
& \bigodot_{\mathsf{i}=1}^{\#\beta-1}(\exists \mathsf{a}, \mathsf{k}.\ \ \mathsf{i} < \mathsf{k} \leq \#\beta \wedge \beta_\mathsf{i} \mapsto \mathsf{a}, \beta_\mathsf{k}).
\end{aligned}$$

Trees (Binary)

$\tau \in \text{Trees}$ iff

$$\tau \in \text{Atoms}$$

$$\text{or } \tau = (\tau_1, \tau_2) \text{ where } \tau_1, \tau_2 \in \text{Trees}.$$

Tree Representation (without sharing)

For $\tau \in \text{Trees}$, we define the assertion

$$\mathsf{tree}\,\tau\,(i)$$

by structural induction:

$$\mathsf{tree}\,a\,(i) \text{ iff } \mathbf{emptyheap} \wedge i = a$$

$$\begin{aligned}&\mathsf{tree}\,(\tau_1, \tau_2)\,(i) \text{ iff}\\ &\quad \exists i_1, i_2.\ \ i \mapsto i_1, i_2 \ * \ \mathsf{tree}\,\tau_1\,(i_1) \ * \ \mathsf{tree}\,\tau_2\,(i_2).\end{aligned}$$

Tree Representation (with sharing)

For $\tau \in \text{Trees}$, we define

$$\mathsf{stree}\,\tau\,(i)$$

by:

$$\mathsf{stree}\,a\,(i) \text{ iff } i = a$$

$$\begin{aligned}&\mathsf{stree}\,(\tau_1, \tau_2)\,(i) \text{ iff}\\ &\quad \exists i_1, i_2.\ \ i \mapsto i_1, i_2 \ * \ (\mathsf{stree}\,\tau_1\,(i_1) \wedge \mathsf{stree}\,\tau_2\,(i_2)).\end{aligned}$$

A Problem

Suppose we wish to prove that a recursively defined procedure $\mathsf{copytree}$ satisfies

$$\{\mathsf{stree}\,\tau(\mathsf{i})\}$$
$$\mathsf{copytree}(\mathsf{i};\mathsf{j})$$
$$\{\mathsf{stree}\,\tau(\mathsf{i}) \ * \ \mathsf{tree}\,\tau(\mathsf{j})\}$$

Then, we must use this specification as a hypothesis in proving that the recursive calls in the procedure body satisfy:

$$\{\mathsf{stree}\,\tau_1(\mathsf{i}_1) \wedge \mathsf{stree}\,\tau_2(\mathsf{i}_2)\}$$
$$\mathsf{copytree}(\mathsf{i}_1;\mathsf{j}_1)$$
$$\{(\mathsf{stree}\,\tau_1(\mathsf{i}_1) \wedge \mathsf{stree}\,\tau_2(\mathsf{i}_2)) \ * \ \mathsf{tree}\,\tau_1(\mathsf{j}_1)\}$$
$$\mathsf{copytree}(\mathsf{i}_2;\mathsf{j}_2)$$
$$\{(\mathsf{stree}\,\tau_1(\mathsf{i}_1) \wedge \mathsf{stree}\,\tau_2(\mathsf{i}_2)) \ * \ \mathsf{tree}\,\tau_1(\mathsf{j}_1) \ * \ \mathsf{tree}\,\tau_2(\mathsf{j}_2)\}$$

But the hypothesis is not strong enough to imply this.

Using Assertion Variables

The procedure

$$
\begin{array}{l}
\mathsf{copytree}(\mathsf{i};\mathsf{j}) = \\
\quad \textbf{if } \mathsf{isatom}(\mathsf{i}) \textbf{ then } \mathsf{j} := \mathsf{i} \textbf{ else} \\
\quad\quad \textbf{newvar } \mathsf{i}_1, \mathsf{i}_2, \mathsf{j}_1, \mathsf{j}_2 \textbf{ in} \\
\quad\quad\quad \big(\mathsf{i}_1 := [\mathsf{i}] \,;\, \mathsf{i}_2 := [\mathsf{i}+1] \,; \\
\quad\quad\quad \textbf{copytree}(\mathsf{i}_1;\mathsf{j}_1) \,;\, \textbf{copytree}(\mathsf{i}_2;\mathsf{j}_2) \,;\, \mathsf{j} := \textbf{cons}(\mathsf{j}_1,\mathsf{j}_2)\big)
\end{array}
$$

satisfies

$$
\begin{array}{l}
\{\mathsf{p} \wedge \mathsf{stree}\,\tau(\mathsf{i})\} \\
\mathsf{copytree}(\mathsf{i};\mathsf{j}) \\
\{\mathsf{p} \,*\, \mathsf{tree}\,\tau(\mathsf{j})\}
\end{array}
$$

since

$$
\begin{array}{l}
\{\mathsf{p} \wedge \mathsf{stree}\,\tau(\mathsf{i})\} \\
\textbf{if } \mathsf{isatom}(\mathsf{i}) \textbf{ then} \\
\quad \{\mathsf{p} \wedge \mathsf{isatom}(\tau) \wedge \tau = \mathsf{i}\} \\
\quad \{\mathsf{p} \ast (\mathsf{isatom}(\tau) \wedge \tau = \mathsf{i} \wedge \mathbf{emptyheap})\} \\
\quad \mathsf{j} := \mathsf{i} \\
\quad \{\mathsf{p} \ast (\mathsf{isatom}(\tau) \wedge \tau = \mathsf{j} \wedge \mathbf{emptyheap})\} \\
\textbf{else} \\
\quad \{\exists \tau_1, \tau_2.\ \tau = (\tau_1, \tau_2) \wedge \mathsf{p} \wedge \mathsf{stree}\,(\tau_1, \tau_2)(\mathsf{i})\} \\
\quad \textbf{newvar } \mathsf{i}_1, \mathsf{i}_2, \mathsf{j}_1, \mathsf{j}_2 \textbf{ in} \\
\qquad \big(\mathsf{i}_1 := [\mathsf{i}]\,;\, \mathsf{i}_2 := [\mathsf{i} + 1]\,; \\
\qquad \{\exists \tau_1, \tau_2.\ \tau = (\tau_1, \tau_2) \wedge \\
\qquad\quad \mathsf{p} \wedge (\mathsf{i} \mapsto \mathsf{i}_1, \mathsf{i}_2 \ast (\mathsf{stree}\,\tau_1\,(\mathsf{i}_1) \wedge \mathsf{stree}\,\tau_2\,(\mathsf{i}_2)))\} \\
\qquad \{\exists \tau_1, \tau_2.\ \tau = (\tau_1, \tau_2) \wedge \mathsf{p} \wedge \mathsf{stree}\,\tau_1\,(\mathsf{i}_1) \wedge \mathsf{stree}\,\tau_2\,(\mathsf{i}_2)\} \\
\qquad \mathbf{copytree}(\mathsf{i}_1; \mathsf{j}_1)\,; \\
\qquad \{\exists \tau_1, \tau_2.\ (\tau = (\tau_1, \tau_2) \wedge \mathsf{p} \wedge \mathsf{stree}\,\tau_2\,(\mathsf{i}_2)) \ast \mathsf{tree}\,\tau_1\,(\mathsf{j}_1)\} \\
\qquad \mathbf{copytree}(\mathsf{i}_2; \mathsf{j}_2)\,; \\
\qquad \{\exists \tau_1, \tau_2.\ (\tau = (\tau_1, \tau_2) \wedge \mathsf{p}) \ast \mathsf{tree}\,\tau_1\,(\mathsf{j}_1) \ast \mathsf{tree}\,\tau_2\,(\mathsf{j}_2)\} \\
\qquad \mathsf{j} := \mathbf{cons}(\mathsf{j}_1, \mathsf{j}_2) \\
\qquad \{\exists \tau_1, \tau_2.\ (\tau = (\tau_1, \tau_2) \wedge \mathsf{p}) \ast \\
\qquad\quad \mathsf{j} \mapsto \mathsf{j}_1, \mathsf{j}_2 \ast \mathsf{tree}\,\tau_1\,(\mathsf{j}_1) \ast \mathsf{tree}\,\tau_2\,(\mathsf{j}_2)\} \\
\qquad \{\exists \tau_1, \tau_2.\ (\tau = (\tau_1, \tau_2) \wedge \mathsf{p}) \ast \mathsf{tree}\,(\tau_1, \tau_2)\,(\mathsf{j})\}\big) \\
\{(\mathsf{p} \ast \mathsf{tree}\,\tau(\mathsf{j})\}
\end{array}
$$

Future Directions

- Higher-order logic and syntactic control of interference
- Concurrency (and passivity)
- Regions
- Axiomatizing directed graphs
- Garbage collection
- Connections with type systems - relative pointers
- Embedding code pointers in data structures