Local Reasoning about

Programs that Alter Data Structure

Peter O'Hearn
Queen Mary, Univ of London

John Reynolds
Carnegie Mellon University

Hongseok Yang
Univ of Illinois and Univ of Birmingham

Slides for talk at CSL'01

References

1. P. W. O'Hearn and D. J. Pym. The Logic of Bunched Implications. *Bulletin of Symbolic Logic* 5(2):215–244. June 1999.

2. J. C. Reynolds. Intuitionistic Reasoning about Shared Mutable Data Structure. *Millenial Perspectives in Computer Science: Proceedings of the 1999 Oxford–Microsoft Symposium in Honour of Sir Tony Hoare.* Palgrove, 2000.

3. S. Ishtiaq and P. W. O'Hearn. BI as an assertion language for mutable data structures. *Conference Record of the Twenty-Eighth Annual ACM Symposium on Principles of Programming Languages*. London, January 2001.

4. R. M. Burstall. Some techniques for proving correctness of programs which alter data structures. Machine Intelligence 7, 1972.

The Model

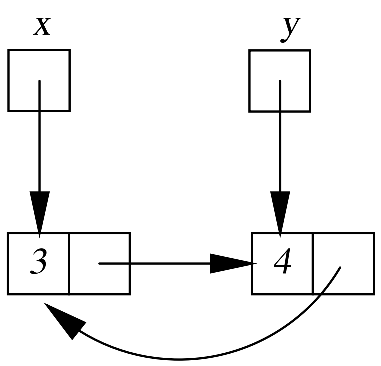
Domain Equations (set eqns)

$$\begin{array}{l} \mathtt{Loc} \subseteq \mathtt{Int} \qquad \mathtt{nil} \in \mathtt{Int} - \mathtt{Loc} \\ S \;=\; \mathtt{Var} \to \mathtt{Int} \\ H \;=\; \mathtt{Loc} \rightharpoonup_f \mathtt{Int} \end{array}$$

Some heap commands. (E, F integer expressions, no $[\cdot]$)

$x := [E]$	(lookup in heap)
$[E] := F$	(alter heap)
$x :=$ *cons*$(E_1, ..., E_n)$	(allocate)
dispose(E)	(deallocate)

$x :=$ *cons*$(3, 3)$; $y :=$ *cons*$(4, 4)$; $[x + 1] := y$; $[y + 1] := x$



Assertions

$$
\begin{array}{lll}
s, h \models E = F & \text{iff} & [\![E]\!]s = [\![F]\!]s \\[1em]
s, h \models E \mapsto F & \text{iff} & \{[\![E]\!]s\} = dom(h) \\
 & & \text{and } h([\![E]\!]s) = [\![F]\!]s \\[2em]
s, h \models \mathtt{emp} & \text{iff} & h = [\,] \text{ is the empty heap} \\[1em]
s, h \models P * Q & \text{iff} & \exists h_0, h_1.\ h_0 \# h_1,\ \ h_0 * h_1 = h, \\
 & & \quad s, h_0 \models P \ \text{ and } \ s, h_1 \models Q \\[1em]
s, h \models P -\!\!* \ Q & \text{iff} & \forall h'. \text{ if } h' \# h \text{ and } s, h' \models P \text{ then} \\
 & & \qquad s, h * h' \models Q \\[2em]
s, h \models \mathtt{false} & & \text{never} \\[1em]
s, h \models P \Rightarrow Q & \text{iff} & \text{if } s, h \models P \ \text{ then } \ s, h \models Q \\[1em]
s, h \models \exists x.P & \text{iff} & \exists v \in \mathtt{Int}.\ [s \mid x \mapsto v], h \models P
\end{array}
$$

- $dom(h)$ denotes the domain of a heap $h \in H$;
- $h \# h'$ iff $dom(h) \cap dom(h') = \emptyset$;
- $h * h'$ denotes the union of disjoint heaps (partial operation)

The Logic of Bunched Implications

BI Algebra: a single poset with: (i) a Heyting algebra structure; and (ii) another (perhaps different) residuated commutative monoid (smcc) structure structure.

$$P * Q \leq R \text{ iff } P \leq Q \mathrel{-\!\!*} R$$

$$P \wedge Q \leq R \text{ iff } P \leq Q \Rightarrow R$$

A *Boolean BI Algebra* is a BI algebra where the Heyting component is boolean.

Observation (Lambek, 60s): The subsets of a (commutative) monoid form a residuated monoid.

This is used in the semantics of many substructural logics.

Older Observation: subsets form a boolean algebra, too

BI accepts older, along with newer. Simple-minded additives.

Here, no Weakening or Contraction

$$\begin{array}{rcl} (x \mapsto 1) * (y \mapsto 2) & \not\Rightarrow & (x \mapsto 1) \\ (x \mapsto 1) & \not\Rightarrow & (x \mapsto 1) * (x \mapsto 1) \end{array}$$

Some Abbreviations

$$
\begin{aligned}
E \mapsto - \;&\triangleq\; \exists y.\, E \mapsto y \qquad (y \notin \mathtt{free}(E)) \\
E \hookrightarrow F \;&\triangleq\; E \mapsto F \;*\; \mathtt{true} \\
E \mapsto E_0, ..., E_n \;&\triangleq\; (E \mapsto E_0) * \cdots * (E + n \mapsto E_n)
\end{aligned}
$$

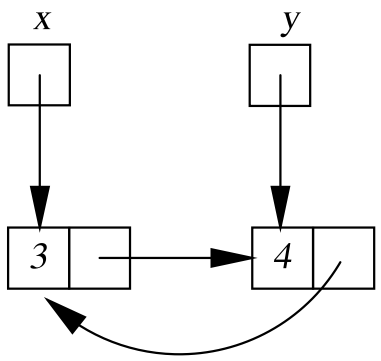
Splitting the heap

$s, h \models x \mapsto E, F$ holds just if

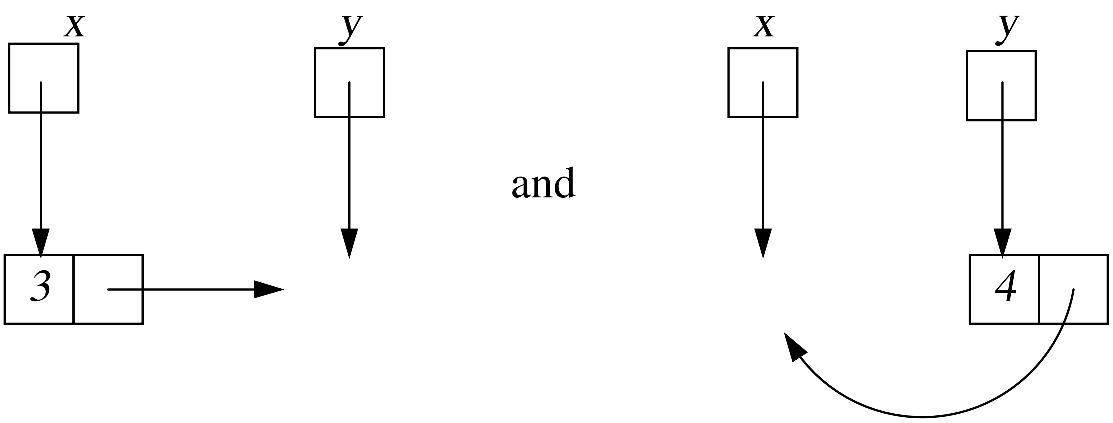
$$\{sx, sx + 1\} = dom(h) \text{ and}$$

$$h(sx) = [\![E]\!]s \text{ and } h(sx + 1) = [\![F]\!]s$$

$(x \mapsto 3, y) * (y \mapsto 4, x)$ is satisfied by



Splitting the heap results in



A Circular Offset-list

We use arrows in the usual way in diagrams, but in the model (and the machine) they are just numbers.

For example, an offset list stores an offset rather than a direct pointer in its link field.

Here is a formula that describes a two-element, circular offset list, with a and b in its head nodes.

$$(x \mapsto a, o) * (x + o \mapsto b, -o)$$

In a store where $x = 17$ and $o = 25$, this is true of

17	a	42	b
18	25	43	-25

Three "Small" Axioms

$$\{E \mapsto -\}\, [E] := b\, \{E \mapsto b\}$$

$$\{\texttt{emp}\}\, x := cons(y, z)\, \{x \mapsto y, z\}$$

$$\{E \mapsto -\}\, dispose(E)\, \{\texttt{emp}\}$$

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}\ \text{ModifiesOnly}(C) \cap \text{free}(R) = \emptyset$$

- ModifiesOnly$(x := -) = \{x\}$
- ModifiesOnly$([E] := F) = \{\}$
- ModifiesOnly$(dispose(E)) = \{\}$.

Warm up

$$\dfrac{\dfrac{\{x \mapsto a\}\,[x] := b\,\{x \mapsto b\}}{\{x \mapsto a * x + 1 \mapsto c\}[x] := b\{x \mapsto b * x + 1 \mapsto c\}}}{\{x \mapsto a, c\}\,[x] := b\,\{x \mapsto b, c\}}$$

Similarly,

$$\{x \mapsto a, c\}\,[x + 1] := b\,\{x \mapsto a, b\}$$

We can reason about components of a cons cell or record in isolation from one another.

On Allocation

Suppose $x \notin \mathrm{free}(P)$.

$$\dfrac{\dfrac{\{\texttt{emp}\}\, x := cons(3,4)\, \{x \mapsto 3,4\,\}}{\{\texttt{emp} * P\}\, x := cons(3,4)\, \{(x \mapsto 3,4)\; * P\}}\, Frame}{\{P\}\, x := cons(3,4)\, \{(x \mapsto 3,4)\; * P\}}\, Consequence$$

Variant handles $x \in \mathrm{free}(P)$ Inserting into middle of linked list

$$\begin{array}{l}
\{(x \mapsto a, y) * (y \mapsto c, w)\} \\
t := \textit{cons}(b, y) \\
\{(t \mapsto b, y) * (x \mapsto a, y) * (y \mapsto c, w)\} \\
\{(x \mapsto a, y) * (t \mapsto b, y) * (y \mapsto c, w)\} \\
[x + 1] := t \\
\{(x \mapsto a, t) * (t \mapsto b, y) * (y \mapsto c, w)\}
\end{array}$$

On Disposal

$$\cfrac{\cfrac{\{E \mapsto -\}\, dispose(E)\, \{\mathtt{emp}\}}{\{(E \mapsto -) * P\}\, dispose(E)\, \{\mathtt{emp} * P\}}\, Frame}{\{(E \mapsto -) * P\}\, dispose(E)\, \{P\}}\, Consequence$$

This is the weakest precondition.

Deleting from middle of a linked list

$$\{(x \mapsto a, t) * (t \mapsto b, y) * (y \mapsto c, w)\}$$
$$[x + 1] := y$$
$$\{(x \mapsto a, y) * (t \mapsto b, y) * (y \mapsto c, w)\}$$
$$\{(t \mapsto b, y) * (x \mapsto a, y) * (y \mapsto c, w)\}$$
dispose(t) ; *dispose*$(t + 1)$
$$\{(x \mapsto a, y) * (y \mapsto c, w)\}$$

More Weakest Preconditions

$$\frac{\{\texttt{emp}\}\, x := cons(3,4)\, \{x \mapsto 3, 4\,\}}{\begin{array}{l}\{\texttt{emp} * (\forall x.\, (x \mapsto 3,4) \mathrel{-\!\!*} P)\}\\ x := cons(3,4)\\ \{x \mapsto 3,4\, * (\forall x.\, (x \mapsto 3,4) \mathrel{-\!\!*} P)\}\end{array}}\, Frame$$

$$\frac{}{\{(\forall x.\, (x \mapsto 3,4) \mathrel{-\!\!*} P)\}\, x := cons(3,4)\, \{P\}}\, Consequence$$

Weakest Precondition

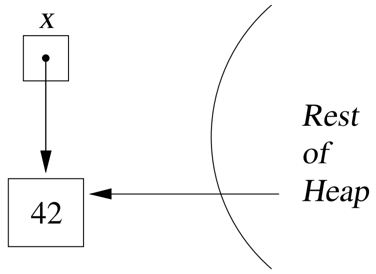
$$\frac{\{x \mapsto -\}\, [x] := 7\, \{x \mapsto 7\}}{\begin{array}{l}\{(x \mapsto -) * ((x \mapsto 7) \mathrel{-\!\!*} P)\}\\ [x] := 7\\ \{(x \mapsto 7) * ((x \mapsto 7) \mathrel{-\!\!*} P)\}\end{array}}\, Frame$$

$$\frac{}{\{(x \mapsto -) * ((x \mapsto 7) \mathrel{-\!\!*} P)\}\, [x] := 7\, \{P\}}\, Consequence$$

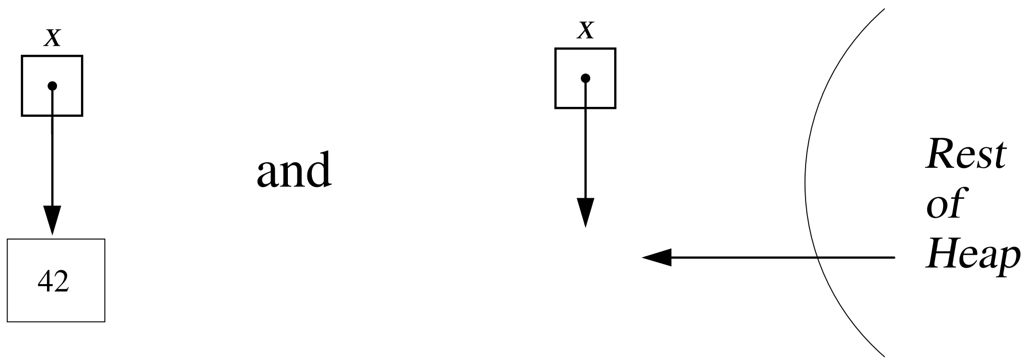
Weakest Precondition.

$*$, $\mathrel{-\!\!*}$ interaction: Update as deletion then extension

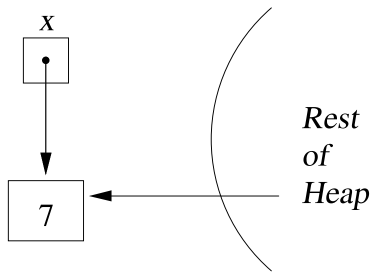
To evaluate $(x \mapsto -) * ((x \mapsto 7) \mathrel{-\!\!*} P)$, split heap, say



into two portions



$\mathrel{-\!\!*}$ ensures P must be true for heap extension



Memory Faults and Tight Specifications

Commands can "go wrong": $dispose(x)$; $[x] := 17$.
Operational semantics with: $C, s, h \triangleright^* \texttt{fault}$

$\{P\}C\{Q\}$ holds iff $\forall s, h.$ if $s, h \models P$ then
$\neg\, C, s, h \triangleright^* \texttt{fault}$
and, if $C, s, h \triangleright^* s', h'$ then $s', h' \models Q$

The avoidance of memory faults in specifications ensures that a well-specified program can only dereference those heap cells guaranteed to exist by the precondition.

Suppose, for example,

$$\{x \mapsto 1, 2\}\, C\, \{x \mapsto 3, 2\}$$

and C only modifies heap, not stack. Now, give C a state with more heap.

$$\{(x \mapsto 1, 2) * (y \mapsto 17, 42)\}\, C\, \{(x \mapsto 3, 2) * ???\}$$

Claim: the heap cell y can't change (in a terminating execution). If it did, the attempt to access it would generate a fault when given a state where y is dangling, contradicting the supposition.

Summary I: Structural Rules

FRAME RULE

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}\,\mathtt{Modifies}(C) \cap \mathtt{Free}(R) = \emptyset$$

AUXILIARY VARIABLE ELIMINATION

$$\frac{\{P\}\,C\,\{Q\}}{\{\exists x.P\}\,C\,\{\exists x.Q\}}\,x \notin \mathtt{Free}(C)$$

AUXILIARY VARIABLE RENAMING

$$\frac{\{P\}\,C\,\{Q\}}{\{P[y/x]\}\,C\,\{Q[y/x]\}}\,x \notin \mathtt{Free}(C),\, y \notin \mathtt{Free}(C, P, Q)$$

CONSEQUENCE

$$\frac{P' \Rightarrow P \qquad \{P\}\,C\,\{Q\} \qquad Q \Rightarrow Q'}{\{P'\}\,C\,\{Q'\}}$$

Summary II: The Small Axioms

$\{E \mapsto -\}\, [E] := F \,\{E \mapsto F\}$

$\{E \mapsto -\}$ *dispose*$(E)\, \{\mathtt{emp}\}$

$\{\mathtt{emp} \wedge x = n\}$
$x := cons(E_1, ..., E_m)$
$\{x \mapsto E_1[n/x], ..., E_m[n/x]\, \}$

$\{x = n \wedge \mathtt{emp}\}\, x := E\, \{x = (E[n/x]) \wedge \mathtt{emp}\}$

$\{E \mapsto n \wedge x = m\}\, x := [E]\, \{x = n \wedge E[m/x] \mapsto n\}$

The rule for *cons* covers fixed-length allocation only. Reynolds has an “iterated independent conjunction” for handling variable length.

$$\begin{array}{l} \{\mathtt{emp}\} \\ x := \mathtt{alloc}(E) \\ \{\bigodot_{i=x}^{x+E-1} i \mapsto -\} \end{array}$$

Can consider alternate expression – $\forall i.\, (i \hookrightarrow -) \Leftrightarrow (x \leq i < x + E)$ – but $\bigodot$ has structural properties.

Example: Tree Disposal

Representation of binary trees

$$\begin{array}{rcl} \textit{tree}\, a\, i & \stackrel{\Delta}{\iff} & i = a \wedge \texttt{isatom?}(a) \wedge \texttt{emp} \\ \textit{tree}\,(\tau_1, \tau_2)\, i & \stackrel{\Delta}{\iff} & \exists x, y.\, (i \mapsto x, y) \, * \, (\textit{tree}\, \tau_1\, x \, * \, \textit{tree}\, \tau_2\, y) \end{array}$$

$\texttt{procedure DispTree}(p)$
$\texttt{newvar}\; i, j;$
$\texttt{if}\;\; \neg\texttt{isatom?}(p)\; \texttt{then}$
$\quad i := [p];\; j := [p+1];$
$\quad \texttt{DispTree}(i);$
$\quad \texttt{DispTree}(j);$
$\quad \textit{dispose}(p);\; \textit{dispose}(p+1)$

The spec

$$\{\textit{tree}\,\tau\, p\}\, \texttt{DispTree}(p; q)\, \{\texttt{emp}\}.$$

Crucial Part of Verification

$\{(p \mapsto x, y) * (\textit{tree}\, \tau_1\, x) * (\textit{tree}\, \tau_2\, y)\}$

$\quad i := [p];\ j := [p + 1];$

$\{(p \mapsto i, j) * (\textit{tree}\, \tau_1\, i) * (\textit{tree}\, \tau_2\, j)\}$

$\quad \texttt{DispTree}(i);$

$\{(p \mapsto i, j) * (\textit{tree}\, \tau_2\, j)\}$

$\quad \texttt{DispTree}(j);$

$\{(p \mapsto i, j)\}$

$\{(p \mapsto i) * (p + 1 \mapsto j)\}$

$\quad \textit{dispose}(p);$

$\{(p + 1 \mapsto j)\}$

$\quad \textit{dispose}(p + 1)$

$\{\texttt{emp}\}$

How do we get 2nd and 3rd steps?

Inferring Frame Axioms

Second recursive call doesn't affect $p \mapsto i, j$... choose $R =\ p \mapsto i, j$ in the Frame Rule

$$\dfrac{\dfrac{\{\textit{tree}\,\tau_2\,j\}\,\texttt{DispTree}(j)\,\{\texttt{emp}\}}{\{(\textit{tree}\,\tau_2\,j) * (p \mapsto i, j)\}\,\texttt{DispTree}(j)\,\{\texttt{emp} * (p \mapsto i, j)\}}}{\{(p \mapsto i, j) * (\textit{tree}\,\tau_2\,j)\}\,\texttt{DispTree}(j)\,\{p \mapsto i, j\}}$$

First recursive call doesn't affect $(p \mapsto i, j)$ or $(\textit{tree}\,\tau_2\,j)$

$$\dfrac{\{\textit{tree}\,\tau_1\,i\}\,\texttt{DispTree}(i)\,\{\texttt{emp}\}}{\begin{array}{l}\{(p \mapsto i, j) * (\textit{tree}\,\tau_1\,i) * (\textit{tree}\,\tau_2\,j)\}\\ \texttt{DispTree}(i)\\ \{(p \mapsto i, j) * (\textit{tree}\,\tau_2\,j)\}\end{array}}$$

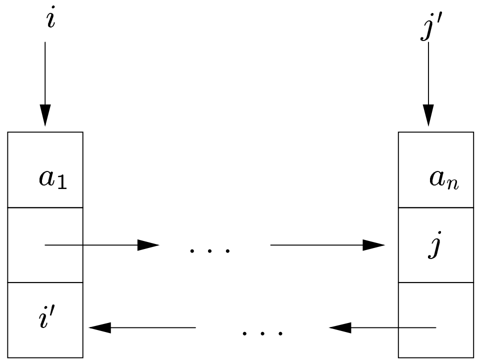
We didn't say " doesn't affect" in the specs.

Without the frame rule, the verification would fail.

Doubly-linked Lists

Conventional doubly-linked lists.

$$\texttt{dl}\, a_1 \cdots a_n\, (i, i', j, j')$$



$$\begin{aligned}
\texttt{dl}\, \epsilon\, (i, i', j, j') &\stackrel{\Delta}{\iff} \texttt{emp} \ \wedge\ i = j \ \wedge\ i' = j' \\
\texttt{dl}\, a\alpha\, (i, i', k, k') &\stackrel{\Delta}{\iff} \exists j. (i \mapsto a, j, i') * \texttt{dl}\, \alpha\, (j, i, k, k')
\end{aligned}$$

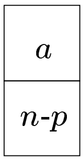
A list terminated by `nil` at both ends is a "total" list

$$\texttt{dl}\, \alpha\, (i, \texttt{nil}, \texttt{nil}, j')$$

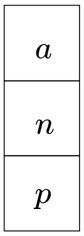
Partial lists result when you pull a total list apart. They are often needed (useful) in verifications.

Difference-linked Lists

For difference-linked lists, we use

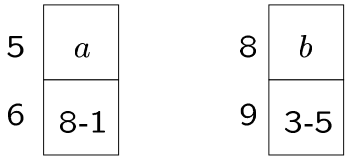


instead of



$$\begin{aligned}
\texttt{dl}\,\epsilon\,(i, i', j, j') &\stackrel{\Delta}{\iff} \texttt{emp} \wedge i = j \wedge i' = j' \\
\texttt{dl}\,a\alpha\,(i, i', k, k') &\stackrel{\Delta}{\iff} \exists j. (i \mapsto a, j - i') * \texttt{dl}\,\alpha\,(j, i, k, k')
\end{aligned}$$

$\texttt{dl}\,ab\,(5, 1, 3, 8)$ is true of



Dequeue

$\{\texttt{dl}\, a\alpha\, (front, \texttt{nil}, \texttt{nil}, back)\}$
$\{\exists n'.\, front \mapsto a, n' - \texttt{nil} \,*\, \texttt{dl}\, \alpha\, (n', front, \texttt{nil}, back)\}$
$\;x := [front];\; d := [front + 1];\; n := d + \texttt{nil};$
$\{x \doteq a \,*\, front \mapsto a, n - \texttt{nil} \,*\, \texttt{dl}\, \alpha\, (n, front, \texttt{nil}, back)\}$
$\;$*dispose*$(front)$; *dispose*$(front + 1)$;
$\{x \doteq a \,*\, \texttt{dl}\, \alpha\, (n, front, \texttt{nil}, back)\}$
$\;\texttt{setlptr}(n, front, \texttt{nil}; back)$
$\{x \doteq a \,*\, \texttt{dl}\, \alpha\, (n, \texttt{nil}, \texttt{nil}, back)\}$

This assumes

$\{\texttt{dl}\, \alpha\, (i, i', \texttt{nil}, j')\}\, \texttt{setlptr}(i, i', k; j')\, \{\texttt{dl}\, \alpha\, (i, k, \texttt{nil}, j')\}$

Swinging a pointer left

The code for $\texttt{setlptr}(i, i', k; j')$ is as follows.

$$\{\texttt{dl}\,\alpha\,(i, i', \texttt{nil}, j')\}$$
$$\texttt{if } i = \texttt{nil} \texttt{ then}$$
$$\{\alpha = \epsilon \wedge \texttt{emp} \wedge i = \texttt{nil}\}$$
$$j' := k$$
$$\{\alpha = \epsilon \wedge \texttt{emp} \wedge i = \texttt{nil} \wedge k = j'\}$$
$$\texttt{else}$$
$$\{\exists \alpha', a, n.\, (\alpha \doteq a\alpha') * \texttt{dl}\,\alpha'\,(n, i, \texttt{nil}, j') * (i \mapsto a, n - i')\}$$
$$[i + 1] := [i + 1] + i' - k$$
$$\{\exists \alpha', a, n.\, (\alpha \doteq a\alpha') * \texttt{dl}\,\alpha'\,(n, i, \texttt{nil}, j') * (i \mapsto a, n - k)\}$$
$$\{\texttt{dl}\,\alpha\,(i, k, \texttt{nil}, j')\}$$

Enqueue

$\{\mathtt{dl}\ \alpha\,(front, \mathtt{nil}, \mathtt{nil}, back)\}$

$\ t := back;$

$\{\mathtt{dl}\ \alpha\,(front, \mathtt{nil}, \mathtt{nil}, t)\}$

$\ back :=$ *cons*$(a, \mathtt{nil} - t);$

$\{\mathtt{dl}\ \alpha\,(front, \mathtt{nil}, \mathtt{nil}, t) \,*\, back \mapsto a, \mathtt{nil} - t\}$

$\ \mathtt{setrptr}(\mathtt{nil}, t, back; front)$

$\{\mathtt{dl}\ \alpha\,(front, \mathtt{nil}, back, t) \,*\, back \mapsto a, \mathtt{nil} - t\}$

$\{\mathtt{dl}\ \alpha a\,(front, \mathtt{nil}, \mathtt{nil}, back)\}$

This assumes

$\{\mathtt{dl}\ \alpha\,(i, \mathtt{nil}, j, j')\}\,\mathtt{setrptr}(j, j', k; i)\{\mathtt{dl}\ \alpha\,(i, \mathtt{nil}, k, j')\}.$

The Power of $-\!\!*$ and Framing

Suppose $\{P\}C\{Q\}$ $\texttt{ModifiesOnly}(C) = X$, and we have a putative postcondition S.

We can use Frame with $R = \forall X.\, Q \mathbin{-\!\!*} S$, followed by two uses of the rule of consequence.

$$
\begin{array}{c}
\{P\}\, C\, \{Q\} \\
\hline
\{P * (\forall X.\, Q \mathbin{-\!\!*} S)\}\, C\, \{Q * (\forall X.\, Q \mathbin{-\!\!*} S) \\
\hline
\{P * (\forall X.\, Q \mathbin{-\!\!*} S)\}\, C\, \{Q * (Q \mathbin{-\!\!*} S)\} \\
\hline
\{P * (\forall X.\, Q \mathbin{-\!\!*} S)\}\, C\, \{S\}
\end{array}
$$

This is the key step in a completeness argument for reasoning from specifications (cf specification statements, refinement calculi).

$-\!\!*$ is powerful, but can often be avoided. Yang uses it in Schorr-Waite proof.

Current Directions

Code pointers

Concurrency

Verification Tool

Abstracting from pointers, AI Frame Problem