Class Notes for CS 819 C - Spring 2002

1. An Introduction to Separation Logic

John C. Reynolds
Department of Computer Science
Carnegie Mellon University

revised January 18, 2002

Two Worlds of Programming

- In low-level languages (C, machine language)
  - \+ minimal constraints on programmer
  - \+ high efficiency possible
  - \+ low-level view of storage and other resources
  - – little protection against errors
  - – platform dependencies
- In strongly typed languages (SML, Java)
  - \+ standard safety properties insured by compilation
  - \+ many errors caught during compilation
  - \+ garbage collection available
  - – correctness not guaranteed
  - – unnecessary run-time checks, e.g.
    - ∗ array-bound checks
    - ∗ double checks for equal-length lists
  - – garbage collection imposed
  - – unnecessarily redundant data structures
  - – obscures low-level view

The Alternative: Formal Specification and Verification of Low-level Programs

- \+ insures both standard safety properties and particular specified properties
- \+ no unnecessary run-time checks
- \+ garbage collection not imposed
- \+ redundant data structures not imposed
- \+ low-level view of storage and other resources
- − not checkable without human intervention

Two Problem Areas	Proposed Solutions
Shared Mutable Data Structures	Separation Logic
Embedded Code Pointers	Reflection of Code into Assertions

The Problem — An Example

A program for in-place list reversal:

$\mathsf{j} := \mathbf{nil}\ ; \mathbf{while}\ \mathsf{i} \neq \mathbf{nil}\ \mathbf{do}\ (\mathsf{k} := [\mathsf{i}+1]\ ; [\mathsf{i}+1] := \mathsf{j}\ ; \mathsf{j} := \mathsf{i}\ ; \mathsf{i} := \mathsf{k}).$

An inadequate invariant:

$$\exists \alpha, \beta.\ \mathsf{list}\ \alpha\ \mathsf{i} \wedge \mathsf{list}\ \beta\ \mathsf{j} \wedge \alpha_0^\dagger = \alpha^\dagger {\cdot} \beta,$$

where

$$\mathsf{list}\ \epsilon\ \mathsf{i} \stackrel{\mathrm{def}}{=} \mathsf{i} = \mathbf{nil} \qquad\qquad \mathsf{list}(\mathsf{a}{\cdot}\alpha)\ \mathsf{i} \stackrel{\mathrm{def}}{=} \exists \mathsf{j}.\ \mathsf{i} \mapsto \mathsf{a}, \mathsf{j} \wedge \mathsf{list}\ \alpha\ \mathsf{j}.$$

An adequate invariant:

$$\begin{aligned}&(\exists \alpha, \beta.\ \mathsf{list}\ \alpha\ \mathsf{i} \wedge \mathsf{list}\ \beta\ \mathsf{j} \wedge \alpha_0^\dagger = \alpha^\dagger {\cdot} \beta)\\ &\quad \wedge (\forall \mathsf{k}.\ \mathbf{reachable}(\mathsf{i}, \mathsf{k}) \wedge \mathbf{reachable}(\mathsf{j}, \mathsf{k}) \Rightarrow \mathsf{k} = \mathbf{nil}),\end{aligned}$$

where

$$\begin{aligned}\mathbf{reachable}(\mathsf{i}, \mathsf{j}) &\stackrel{\mathrm{def}}{=} \exists n \geq 0.\ \mathbf{reachable}_n(\mathsf{i}, \mathsf{j})\\ \mathbf{reachable}_0(\mathsf{i}, \mathsf{j}) &\stackrel{\mathrm{def}}{=} \mathsf{i} = \mathsf{j}\\ \mathbf{reachable}_{n+1}(\mathsf{i}, \mathsf{j}) &\stackrel{\mathrm{def}}{=} \exists \mathsf{a}, \mathsf{k}.\ \mathsf{i} \mapsto \mathsf{a}, \mathsf{k} \wedge \mathbf{reachable}_n(\mathsf{k}, \mathsf{j}).\end{aligned}$$

In separation logic:

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

The Problem (continued)

A stronger invariant:

$$
\begin{aligned}
&(\exists \alpha, \beta.\ \mathsf{list}\,\alpha\,\mathsf{i} \wedge \mathsf{list}\,\beta\,\mathsf{j} \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta)\\
&\quad \wedge (\forall \mathsf{k}.\ \mathbf{reachable}(\mathsf{i},\mathsf{k}) \wedge \mathbf{reachable}(\mathsf{j},\mathsf{k}) \Rightarrow \mathsf{k} = \mathbf{nil})\\
&\quad \wedge \mathsf{list}\,\gamma\,\mathsf{x}\\
&\quad \wedge (\forall \mathsf{k}.\ \mathbf{reachable}(\mathsf{x},\mathsf{k})\\
&\qquad\qquad \wedge (\mathbf{reachable}(\mathsf{i},\mathsf{k}) \vee \mathbf{reachable}(\mathsf{j},\mathsf{k})) \Rightarrow \mathsf{k} = \mathbf{nil}).
\end{aligned}
$$

In separation logic:

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

Overview of Separation Logic

- Low-level programming language
  - Extension of simple imperative language
  - Commands for allocating, accessing, mutating, and deallocating data structures
  - Dangling pointer faults (if pointer is dereferenced)
- Program specification and proof
  - Extension of Hoare logic
  - Separating (independent, spatial) conjunction ($*$)
- Inductive definitions over abstract structures

The Programming Language

The simple imperative language:

$$:= \qquad \textbf{skip} \qquad ; \qquad \textbf{if} - \textbf{then} - \textbf{else} - \qquad \textbf{while} - \textbf{do} -$$

plus:

$$\begin{array}{lll}
 & & \text{Store}: \ \mathsf{x}\colon 3,\ \mathsf{y}\colon 4 \\
 & & \text{Heap}: \ \text{empty} \\
\text{Allocation} & \mathsf{x} := \textbf{cons}(1,2)\ ; & \\
 & & \text{Store}: \ \mathsf{x}\colon 37,\ \mathsf{y}\colon 4 \\
 & & \text{Heap}: \ 37\colon 1,\ 38\colon 2 \\
\text{Lookup} & \mathsf{y} := [\mathsf{x}]\ ; & \\
 & & \text{Store}: \ \mathsf{x}\colon 37,\ \mathsf{y}\colon 1 \\
 & & \text{Heap}: \ 37\colon 1,\ 38\colon 2 \\
\text{Mutation} & [\mathsf{x}+1] := 3\ ; & \\
 & & \text{Store}: \ \mathsf{x}\colon 37,\ \mathsf{y}\colon 1 \\
 & & \text{Heap}: \ 37\colon 1,\ 38\colon 3 \\
\text{Deallocation} & \textbf{dispose}(\mathsf{x}+1) & \\
 & & \text{Store}: \ \mathsf{x}\colon 37,\ \mathsf{y}\colon 1 \\
 & & \text{Heap}: \ 37\colon 1
\end{array}$$

Note that:

- States = Stores × Heaps.
- Expressions depend only upon the store.
- Allocation is nondeterminate.
- Pointers are integers — pointer arithmetic is permitted.

Memory Faults

$$
\begin{array}{lll}
 & & \text{Store}: \ \mathsf{x}\!:3,\ \mathsf{y}\!:4 \\
 & & \text{Heap}: \ \text{empty} \\
\text{Allocation} & \mathsf{x} := \mathbf{cons}(1, 2)\ ; & \\
 & & \text{Store}: \ \mathsf{x}\!:37,\ \mathsf{y}\!:4 \\
 & & \text{Heap}: \ 37\!:1,\ 38\!:2 \\
\text{Lookup} & \mathsf{y} := [\mathsf{x}]\ ; & \\
 & & \text{Store}: \ \mathsf{x}\!:37,\ \mathsf{y}\!:1 \\
 & & \text{Heap}: \ 37\!:1,\ 38\!:2 \\
\text{Mutation} & [\mathsf{x} + 2] := 3\ ; & \\
 & & \mathbf{abort}
\end{array}
$$

Faults can also be caused by out-of-range lookup or deallocation.

Assertions

Standard predicate calculus:

$$\wedge \qquad \vee \qquad \neg \qquad \Rightarrow \qquad \forall \qquad \exists$$

plus:

- **emp**
  The heap is empty.

- $e \mapsto e'$
  The heap contains one cell, at address e with contents e'.

- $p_1 \ * \ p_2$
  The heap can be split into two disjoint parts such that p_1 holds for one part and p_2 holds for the other.

Some Abbreviations

$$e \mapsto - \stackrel{\text{def}}{=} \exists x'.\, e \mapsto x' \quad \text{where } x' \text{ not free in } e$$

$$e \hookrightarrow e' \stackrel{\text{def}}{=} e \mapsto e' \ast \mathbf{true}$$

$$e \mapsto e_0, \ldots, e_{n-1} \stackrel{\text{def}}{=} e \mapsto e_0 \ast \cdots \ast e + n - 1 \mapsto e_{n-1}$$

$$e \hookrightarrow e_0, \ldots, e_{n-1} \stackrel{\text{def}}{=} e \hookrightarrow e_0 \ast \cdots \ast e + n - 1 \hookrightarrow e_{n-1}$$

$$\text{iff } e \mapsto e_0, \ldots, e_{n-1} \ast \mathbf{true}$$

Examples

$\mathsf{x} \mapsto 3, \mathsf{y}$	Store : $\mathsf{x}\colon\alpha,\ \mathsf{y}\colon\beta$ Heap : $\alpha\colon 3,\ \alpha{+}1\colon\beta$
$\mathsf{y} \mapsto 3, \mathsf{x}$	Store : $\mathsf{x}\colon\alpha,\ \mathsf{y}\colon\beta$ Heap : $\beta\colon 3,\ \beta{+}1\colon\alpha$
$\mathsf{x} \mapsto 3, \mathsf{y}\ *\ \mathsf{y} \mapsto 3, \mathsf{x}$	Store : $\mathsf{x}\colon\alpha,\ \mathsf{y}\colon\beta$ Heap : $\alpha\colon 3,\ \alpha{+}1\colon\beta,\ \beta\colon 3,\ \beta{+}1\colon\alpha$ where α, $\alpha + 1$, β, $\beta + 1$ are distinct
$\mathsf{x} \mapsto 3, \mathsf{y} \wedge \mathsf{y} \mapsto 3, \mathsf{x}$	Store : $\mathsf{x}\colon\alpha,\ \mathsf{y}\colon\alpha$ Heap : $\alpha\colon 3,\ \alpha{+}1\colon\alpha$
$\mathsf{x} \hookrightarrow 3, \mathsf{y} \wedge \mathsf{y} \hookrightarrow 3, \mathsf{x}$	Store : $\mathsf{x}\colon\alpha,\ \mathsf{y}\colon\beta$ Heap : $\alpha\colon 3,\ \alpha{+}1\colon\beta,\ \beta\colon 3,\ \beta{+}1\colon\alpha,\ \ldots$

Specifications

- $\{p\}\ c\ \{q\}$ (partial correctness)

  Starting in any state in which p holds:

  - No execution of c aborts.
  - When some execution of c terminates in a final state, then q holds in the final state.

- $[\,p\,]\ c\ [\,q\,]$ (total correctness)

  Starting in any state in which p holds:

  - No execution of c aborts.
  - Every execution of c terminates.
  - When some execution of c terminates in a final state, then q holds in the final state.

Thus:

• • • Well-specified programs can't go wrong. • • •

— and memory-fault checking is unnecessary.

Violating Record Boundaries

Suppose P holds for some state (s_0, h_0), 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 there is *some* execution such that

$$\begin{tikzcd}
(s_0, h_0) \arrow[d, "C"'] \\
(s_1, h_1) \arrow[d, "\mathsf{x} := \mathbf{cons}(1,2)"'] \\
(s_2, h_2) \text{ where } s_2(\mathsf{x}+2) \notin \operatorname{dom} h \arrow[d, "C'"'] \\
(s_3, h_3) \text{ where } s_3(\mathsf{x}+2) \notin \operatorname{dom} h \arrow[d, "{[\mathsf{x}+2] := 3}"'] \\
\mathbf{abort}
\end{tikzcd}$$

Inference Rules

Mutation

$$\begin{aligned}&\{(e \mapsto -) \ * \ p\}\\ &[e] := e'\\ &\{(e \mapsto e') \ * \ p\}\end{aligned}$$

Allocation

$$\begin{aligned}&\{p\}\\ &v := \mathbf{cons}(e_0, \ldots, e_{n-1})\\ &\{(v \mapsto e_0, \ldots, e_{n-1}) \ * \ p\}\end{aligned}$$

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

Inference Rules (continued)

Lookup

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

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

Deallocation

$$\{(e \mapsto -) \ * \ p\}$$
$$\textbf{dispose}\ e$$
$$\{p\}$$

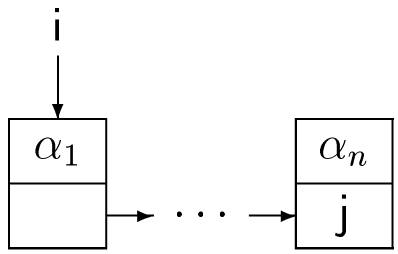
Frame Rule

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

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

Inductive Definitions: Singly-Linked Lists

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



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

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

Properties

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

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

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

Reversing a List in Place

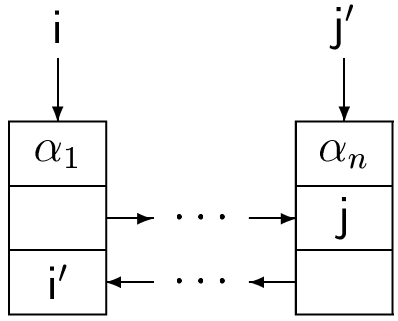
$$
\begin{array}{l}
\{\mathbf{list}\ \gamma\,(\mathsf{i}, \mathbf{nil})\} \\
\mathsf{j} := \mathbf{nil}\ ; \\
\mathbf{while}\ \mathsf{i} \neq \mathbf{nil}\ \mathbf{do} \\
\quad (\mathsf{k} := [\mathsf{i} + 1]\ ;\ [\mathsf{i} + 1] := \mathsf{j}\ ;\ \mathsf{j} := \mathsf{i}\ ;\ \mathsf{i} := \mathsf{k}) \\
\{\mathbf{list}\ \gamma^{\dagger}\,(\mathsf{j}, \mathbf{nil})\}
\end{array}
$$

The Invariant

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

Doubly-Linked Lists

dlist α (i, i′, j, j′):

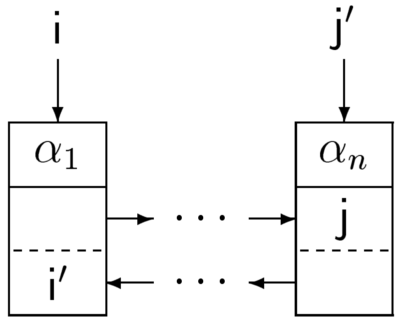


$$\mathsf{dlist}\;\epsilon\,(\mathsf{i},\mathsf{i}',\mathsf{j},\mathsf{j}') \stackrel{\mathrm{def}}{=} \mathbf{emp} \wedge \mathsf{i} = \mathsf{j} \wedge \mathsf{i}' = \mathsf{j}'$$

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

Xor-Linked Lists

xlist α (i, i′, j, j′):



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

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

Binary Trees (S-expressions)

$\tau \in$ Trees iff

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

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

Representation (without sharing)

For $\tau \in$ Trees, we define the assertion

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

by structural induction:

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

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

Representation (with sharing)

For $\tau \in$ Trees, we define

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

by:

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

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

Iterated Separating Conjunction

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

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

Array Allocation

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

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

An Inference Rule

$$\begin{array}{l}
\{p\} \\
v := \textbf{allocate}\ e \\
\{p \,*\, \bigodot_{i=v}^{v+e-1} i \mapsto -\},
\end{array}$$

where v does not occur free in p or e.

Conditional Critical Regions (O'Hearn)

$$
\begin{array}{l}
\mathsf{full} := \mathbf{false}\ ; \\
\mathbf{resource}\ \mathsf{buf}(\mathsf{c}, \mathsf{full})\ \mathbf{in} \\
\mathbf{cobegin} \\
\quad \mathsf{x} := \mathbf{cons}(17)\ ; \\
\quad \mathbf{with}\ \mathsf{buf}\ \mathbf{when}\ \neg\,\mathsf{full}\ \mathbf{do} \\
\qquad (\mathsf{c} := \mathsf{x}\ ; \mathsf{full} := \mathbf{true}) \\
\| \\
\quad \mathbf{with}\ \mathsf{buf}\ \mathbf{when}\ \mathsf{full}\ \mathbf{do} \\
\qquad (\mathsf{y} := \mathsf{c}\ ; \mathsf{full} := \mathbf{false})\ ; \\
\quad \mathbf{dispose}\ \mathsf{y} \\
\mathbf{coend}
\end{array}
$$

New Constructs

- Resource Declaration: $\mathbf{resource}\ r(x_1, \ldots, x_n)\ \mathbf{in}\ \ldots$

where the resource variables $x_1, \ldots, x_n$ can only occur in a conditional critical region keyed to r:

- Conditional Critical Region: $\mathbf{with}\ r\ \mathbf{when}\ B\ \mathbf{do}\ C$

- Concurrent Composition: $\mathbf{cobegin}\ C_1 \parallel \cdots \parallel C_n\ \mathbf{coend}$

where, if a nonresource variable is assigned by C_i, it cannot occur in another C_j.

Conditional Critical Regions (continued)

Let

$$RI \stackrel{\text{def}}{=} (\mathsf{full} \wedge \mathsf{c} \mapsto -) \vee (\neg\,\mathsf{full} \wedge \mathbf{emp}).$$

Then

$$\begin{array}{l}
\{(\mathsf{x} \mapsto - \;*\; RI) \wedge \neg\,\mathsf{full}\} \\
\{\mathsf{x} \mapsto - \;*\; \mathbf{emp}\} \\
\{\mathsf{x} \mapsto -\} \\
\mathsf{c} := \mathsf{x}\,;\, \mathsf{full} := \mathbf{true} \\
\{\mathsf{c} \mapsto - \wedge \mathsf{full}\} \\
\{\mathbf{emp} \;*\; (\mathsf{c} \mapsto - \wedge \mathsf{full})\} \\
\{\mathbf{emp} \;*\; RI\}
\end{array}$$

and

$$\begin{array}{l}
\{(\mathbf{emp} \;*\; RI) \wedge \mathsf{full}\} \\
\{\mathbf{emp} \;*\; \mathsf{c} \mapsto -\} \\
\{\mathsf{c} \mapsto -\} \\
\mathsf{y} := \mathsf{c}\,;\, \mathsf{full} := \mathbf{false} \\
\{\mathsf{y} \mapsto - \wedge \neg\,\mathsf{full}\} \\
\{\mathsf{y} \mapsto - \;*\; (\neg\,\mathsf{full} \wedge \mathbf{emp})\} \\
\{\mathsf{y} \mapsto - \;*\; RI\}.
\end{array}$$

Inference for Conditional Critical Regions

$$\frac{\{(P * RI) \wedge B\}\ C\ \{Q * RI\}}{\{P\}\ \textbf{with}\ r\ \textbf{when}\ B\ \textbf{do}\ C\ \{Q\}}$$

where only the resource variables of r occur in the resource invariant RI of r.

For example

$$\frac{\{(\mathsf{x} \mapsto - * RI) \wedge \neg\,\mathsf{full}\}\ \mathsf{c} := \mathsf{x}\,;\mathsf{full} := \textbf{true}\ \{\textbf{emp} * RI\}}{\{\mathsf{x} \mapsto -\}\ \textbf{with}\ \mathsf{buf}\ \textbf{when}\ \neg\,\mathsf{full}\ \textbf{do}\ (\mathsf{c} := \mathsf{x}\,;\mathsf{full} := \textbf{true})\ \{\textbf{emp}\}}$$

and

$$\frac{\{(\textbf{emp} * RI) \wedge \mathsf{full}\}\ \mathsf{y} := \mathsf{c}\,;\mathsf{full} := \textbf{false}\ \{\mathsf{y} \mapsto - * RI\}}{\{\textbf{emp}\}\ \textbf{with}\ \mathsf{buf}\ \textbf{when}\ \mathsf{full}\ \textbf{do}\ (\mathsf{y} := \mathsf{c}\,;\mathsf{full} := \textbf{false})\ \{\mathsf{y} \mapsto -\}.}$$

Thus

$\{\textbf{emp}\}$
$\mathsf{x} := \textbf{cons}(17)\,;\textbf{with}\ \mathsf{buf}\ \textbf{when}\ \neg\,\mathsf{full}\ \textbf{do}\ (\mathsf{c} := \mathsf{x}\,;\mathsf{full} := \textbf{true})$
$\{\textbf{emp}\}$

and

$\{\textbf{emp}\}$
$\textbf{with}\ \mathsf{buf}\ \textbf{when}\ \mathsf{full}\ \textbf{do}\ (\mathsf{y} := \mathsf{c}\,;\mathsf{full} := \textbf{false})\,;\textbf{dispose}\ \mathsf{y}$
$\{\textbf{emp}\}.$

Inference for Concurrent Composition

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \qquad \{P_2\}\ C_2\ \{Q_2\}}{\{RI\ *\ P_1\ *\ P_2\}\ \textbf{cobegin}\ C_1 \parallel C_2\ \textbf{coend}\ \{RI\ *\ Q_1\ *\ Q_2\}}$$

where no variable free in P_i or Q_i is assigned by another C_j.

For example:

$$\frac{\{\textbf{emp}\}\ C_1\ \{\textbf{emp}\} \qquad \{\textbf{emp}\}\ C_2\ \{\textbf{emp}\}}{\{RI\ *\ \textbf{emp}\ *\ \textbf{emp}\}\ \textbf{cobegin}\ C_1 \parallel C_2\ \textbf{coend}\ \{RI\ *\ \textbf{emp}\ *\ \textbf{emp}\},}$$

where the conclusion simplifies to

$$\{RI\}\ \textbf{cobegin}\ C_1 \parallel C_2\ \textbf{coend}\ \{RI\}.$$

Future Directions

- Axiomatizing directed graphs
- Garbage collection
- Concurrency
- Passivity
- Regions
- Connections with type systems
- Embedding code pointers in data structures