An introduction to separation logic

David Schmidt

**Kansas State University (USA)
and Projet Lande, IRISA (F)**

`www.cis.ksu.edu/~schmidt`

Outline

1. Review Hoare-logic rules for program validation
2. Examine problems related to objects, pointers, and aliasing
3. Introduce graph models of storage
4. Introduce *separation logic* for validating programs that use the graph models
5. If time allows, see how separation logic can be used to prove properties of concurrent programs that share resources

When we write programs, we depend on logical properties

$$
\begin{array}{l}
\texttt{class BankAccount} \; \{ \\
\quad \texttt{private int balance;} \; \{invariant: \texttt{balance} \geq 0\} \\
\quad \ldots \\
\quad \texttt{deposit(int x)}\{ \;\; \{precondition: \texttt{x} > 0\} \\
\qquad \texttt{balance} := \texttt{balance} + \texttt{x}; \\
\quad \} \; \textit{is the invariant preserved? Is } \texttt{balance} \geq 0? \\
\quad \ldots \\
\}
\end{array}
$$

Floyd, Hoare, and Wirth proposed logical laws for programs

Assignment axiom:

$$\{[E/\mathtt{x}]P\}\ \ \mathtt{x} := E\ \ \{P\}$$

where $[E/\mathtt{x}]P$ denotes the substitution of E for all free occurrences of $\mathtt{x}$ in P.

Example:

$$\{\mathtt{balance} + \mathtt{x} \geq 0\}\ \ \mathtt{balance} := \mathtt{balance} + \mathtt{x}\ \ \{\mathtt{balance} \geq 0\}$$

(It helps to read the rule and the example from right to left.)

Since `deposit`*'s precondition asserted that* $\mathtt{x} > 0$*, and* `BankAccount`*'s class invariant said that* $\mathtt{balance} \geq 0$*, we conclude that*

$$\{\mathtt{balance} \geq 0 \land \mathtt{x} > 0\}\ \ \mathtt{balance} := \mathtt{balance} + \mathtt{x}\ \ \{\mathtt{balance} \geq 0\},$$

proving that `deposit` *preserves the class invariant.*

Composition rule for commands

$$\frac{\{P\}S_1\{Q\} \quad \{Q\}S_2\{R\}}{\{P\}S_1; S_2\{R\}}$$

Example: validating the exchange of two values

$$\{x = a \land y = b\}$$

$$\texttt{temp} := y;$$

$$\{x = a \land \texttt{temp} = b\}$$

$$y := x;$$

$$\{y = a \land \texttt{temp} = b\}$$

$$x := \texttt{temp}$$

$$\{y = a \land x = b\}$$

The rules for conditionals and loops

$$\frac{\{E \wedge P\}S_1\{Q\} \quad \{\neg E \wedge P\}S_2\{Q\}}{\{P\}\texttt{if } E \texttt{ then } S_1 \texttt{ else } S_2\{Q\}} \qquad \frac{\{E \wedge P\}S\{P\}}{\{P\}\texttt{while } E \texttt{ do } S\{P \wedge \neg E\}}$$

Example: validating factorial using the *loop invariant*, $\texttt{fac} = \texttt{i}!$

$$\begin{array}{l}
\quad \texttt{i} := 0;\ \texttt{fac} := 1; \\
\{\texttt{fac} = \texttt{i}!\} \\
\quad \texttt{while i} \neq \texttt{x do} \ \{ \\
\quad \{\texttt{i} \neq \texttt{x} \wedge \texttt{fac} = \texttt{i}!\} \\
\qquad \texttt{i} := \texttt{i} + 1;\ \texttt{fac} := \texttt{fac} * \texttt{i} \\
\quad \{\texttt{fac} = \texttt{i}!\} \\
\quad \} \\
\{\texttt{fac} = \texttt{i}! \wedge \texttt{i} = \texttt{x}\} \\
\{\texttt{fac} = \texttt{x}!\}
\end{array}$$

The rules are **unsound** *when aliasing is allowed*

Assignment axiom: $\{[E/\mathtt{x}]P\}\ \mathtt{x} := E\ \{P\}$

Program: `x:= new Cell(3,nil); y:= x; y.head:= 4`

Read the example from the bottom to the top:

$$\{4 > 3\}$$

$$\{4 > \mathtt{new\ Cell(3,nil).head}\}$$

$$\mathtt{x} := \mathtt{new\ Cell(3,nil)}$$

$$\{4 > \mathtt{x.head}\}$$

$$\mathtt{y} := \mathtt{x}$$

$$\{4 > \mathtt{x.head}\}$$

$$\mathtt{y.head} := 4$$

$$\{\mathtt{y.head} > \mathtt{x.head}\}$$

We proved $\mathtt{y.head} > \mathtt{x.head}$, *even though* `x` *and* `y` *point to the same Cell object!*

The aliasing problem also appears when we use procedures with call-by-reference (call-by-location) parameter passing and/or arrays.

Since objects, references ("pointers"), and aliasing are standard features, we must develop a more discriminating semantic model and more discriminating inference rules for programs.

Our semantic model of storage will be a graph structure, speci£cally, a *Kripke structure*.

Storage is a Kripke structure, $G = \langle \Sigma, \tau, \mathcal{I} \rangle$

$\Sigma = \{c0, c1, c2\}$

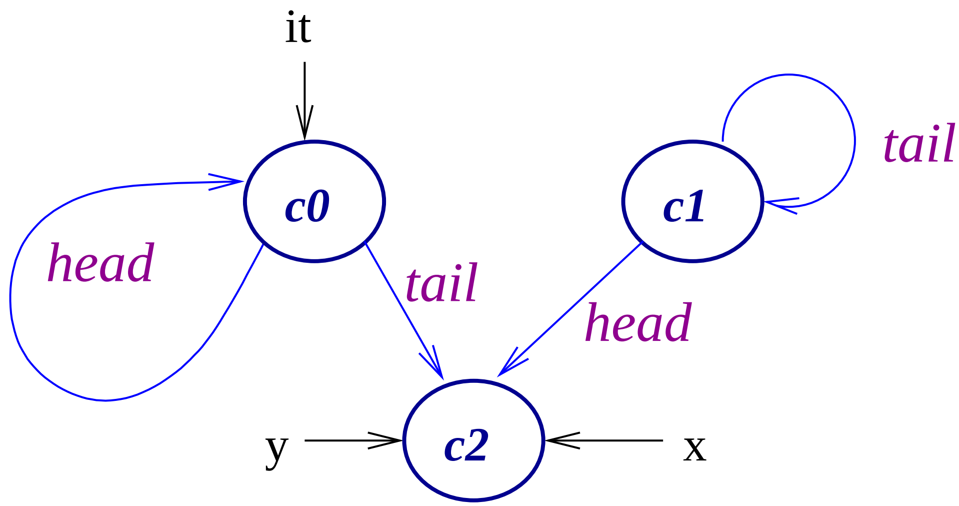
$\tau_{head} = \{(c0, c0)\}$

$\tau_{tail} = \{(c0, c2),$
$\quad (c1, c1), (c1, c2)\}$

$\mathcal{I}(c0) = \{it\}$

$\mathcal{I}(c1) = \{\}$

$\mathcal{I}(c2) = \{x, y\}$



The nodes are *cells* (objects). We express properties of the graph:

$$G \models tail(it, x) \wedge x = y$$

$$G \models \exists n. tail(n, n)$$

We validate graph properties across state changes, $\{P\}\, S\, \{Q\}$*:*

$\{x = y\}$

$\quad \texttt{it.tail} := \texttt{x}$

$\{tail(\texttt{it}, \texttt{x}) \wedge \texttt{x} = \texttt{y}\}$

That is, if $G_{pre} \models P$,
and $G_{post} = [\![S]\!] G_{pre}$,
then $G_{post} \models Q$

Modular validation

A property, ϕ, is checked with respect to the entire graph, G,

$$G \models \phi.$$

Is there a "modular" variant of property checking, where a subgraph of G is used to validate ϕ?

That is, we want to divide the model, G, into disjoint "regions" (subgraphs), h_i, so that we can use this reasoning principle:

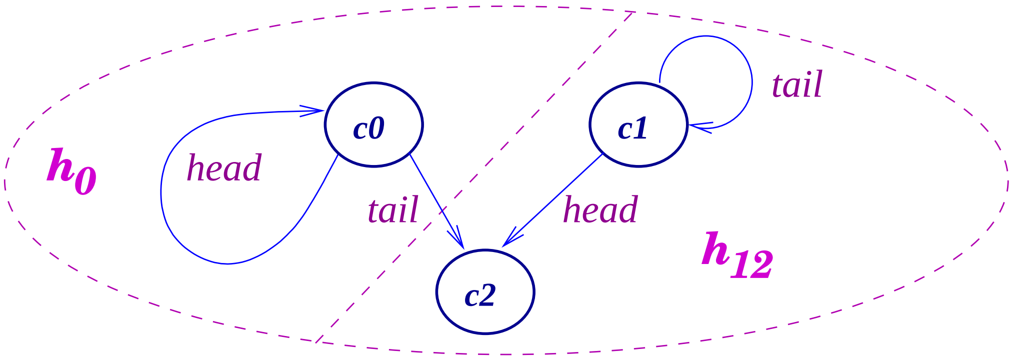
$$\frac{h_1 \models \phi_1 \quad h_2 \models \phi_2 \quad h_1 \# h_2}{h_1 \circ h_2 \models \phi_1 * \phi_2}$$

where $h_1 \# h_2$ asserts that h_1 and h_2 are disjoint regions of G.
$\phi_1 * \phi_2$ expresses an assertion whose conjuncts hold true for disjoint regions — *no aliasing/sharing between regions!*

This is the motivation for *separation logic*.

We will apply separation logic to storage heaps, e.g.,

$h = \langle \mathrm{Cell}, \{\mathrm{head}, \mathrm{tail}\}\rangle$. (Say that $\mathrm{domain}(h) = \mathrm{Cell}$.)



We say that $\langle \mathrm{Cell}_1, \{\mathrm{head}_1, \mathrm{tail}_1\}\rangle \# \langle \mathrm{Cell}_2, \{\mathrm{head}_2, \mathrm{tail}_2\}\rangle$ iff $\mathrm{Cell}_1 \cap \mathrm{Cell}_2 = \{\}$
— *their domains (node sets) are disjoint.*

The composition of two heap-regions is

$$\langle \mathrm{Cell}_1, \{\mathrm{head}_1, \mathrm{tail}_1\}\rangle \circ \langle \mathrm{Cell}_2, \{\mathrm{head}_2, \mathrm{tail}_2\}\rangle$$
$$= \langle \mathrm{Cell}_1 \cup \mathrm{Cell}_2, \{\mathrm{head}_1 \cup \mathrm{head}_2, \mathrm{tail}_1 \cup \mathrm{tail}_2\}\rangle$$

if $\mathrm{Cell}_1 \# \mathrm{Cell}_2$ (else the composition is unde£ned).

The above diagram displays two disjoint regions, h_0 and h_{12}:

$$h_0 = \langle \{c_0\}, \{\{(c_0, c_0)\}, \{(c_0, c_2)\}\}\rangle$$
$$h_{12} = \langle \{c_1, c_2\}, \{\{(c_1, c_2)\}, \{(c_1, c_1)\}\}\rangle$$

h_0 *shows that a graph can contain "dangling edges" (free references).*

2. *Separation logic* *(O'Hearn, Pym, Reynolds, Yang)*

Additives (the logic that expresses graph properties): Let $h \in \mathsf{Heap}$:

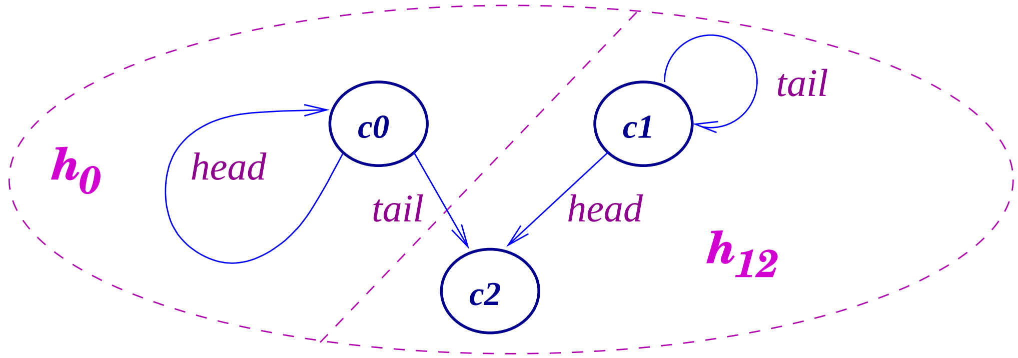
$h \models p \wedge p'$	iff $h \models p$ and $h \models p'$ (similar for $h \models p \vee p'$)
$h \models p \rightarrow p'$	iff $h \models p$ implies $h \models p'$
$h \models \exists x.p_x$	iff exists $v \in \mathsf{Cell}$ s.t. $h \models p_v$ (similar for $h \models \forall x.p_x$)
$h \models \mathit{false}$	never
$h \models R(E_1, E_2)$	... application dependent: see examples that follow

Multiplicatives (based on a commutative partial monoid, $(\mathsf{Heap}, \circ, \epsilon)$):

$h \models \mathit{emp}$	iff $h = \epsilon$
$h \models p * p'$	iff there exist h_0, h_1 such that $h_0 \# h_1$, $h = h_0 \circ h_1$, $h_0 \models p$, and $h_1 \models p'$
$h \models p \mathrel{-\!\!*} p'$	iff for all h', if $h' \# h$ and $h' \models p$, then $h \circ h' \models p'$

Let $\mathrm{h} \models tl(\mathrm{a}, \mathrm{b})$ *iff* $\mathrm{domain}(\mathrm{h}) = \{\mathrm{a}\}$ *and* $(\mathrm{a}, \mathrm{b}) \in tail$. *That is,* h *has one cell,* a, *whose* $tail$ *£eld holds address* b. *(similar for* $\mathrm{h} \models hd(\mathrm{a}, \mathrm{b})$*)*

Examples:



$\mathrm{h}_0 \models tl(\mathrm{c}_0, \mathrm{c}_2)$ Notice the “dangling pointer” (free reference)

$$\mathrm{h}_0 \models \exists \mathrm{c}.\, tl(\mathrm{c}_0, \mathrm{c})$$

$$\mathrm{h}_0 \models tl(\mathrm{c}_0, \mathrm{c}_2) \wedge \exists \mathrm{c}.\, tl(\mathrm{c}_0, \mathrm{c})$$

$$\mathrm{h}_{12} \models \mathrm{c}_1 \neq \mathrm{c}_2$$

$$\mathrm{h}_0 \circ \mathrm{h}_{12} \models (tl(\mathrm{c}_0, \mathrm{c}_2) \wedge \exists \mathrm{c}.\, tl(\mathrm{c}_0, \mathrm{c})) * \mathrm{c}_1 \neq \mathrm{c}_2$$

$$\mathrm{h}_0 \models tl(\mathrm{c}_2, \mathrm{x}) \mathrel{-\!\!*} \exists \mathrm{z}.\, tl(\mathrm{c}_0, \mathrm{z}) \wedge tl(\mathrm{z}, \mathrm{x})$$

Separation logic proves correctness properties

We can write an assertion that de£nes a (tail-)noncircular list:

$$\mathrm{nc}(\ell) \text{ iff}_{lfp} (\ell = \mathrm{null}) \lor (\exists c.\ tl(\ell, c) * \mathrm{nc}(c))$$

The star ($*$) ensures that *all cells in the list's tail live in a region that is disjoint from the one-cell region holding the list's head,* ℓ.

We might prove that a copy function constructs a noncircular list:

```
copy(Cell x)  { precondition:{nc(x)}
  if x = null
      then y:= null; {nc(y)}
      else temp := copy(x.tl); {nc(temp)} %recursion hyp.
           y := new Cell(x.hd, temp) {tl(y,temp) * nc(temp)}
           {nc(y)}
  {nc(y)}
  return y; postcondition:{nc(answer_copy)} }
```

The assignment axiom is replaced by four "small axioms"

First, let $\mathsf{E} \doteq \mathsf{E}'$ abbreviate $\mathsf{E} = \mathsf{E}' \wedge emp$

(where emp asserts that the head is empty: $\epsilon \models emp$).

and let $\mathsf{E}_1 \mapsto \mathsf{E}_2, \mathsf{E}_3$ abbreviate $hd(\mathsf{E}_1, \mathsf{E}_2) \wedge tl(\mathsf{E}_1, \mathsf{E}_3)$.

(Therefore, the heap has exactly one cell, E_1.)

Assume that $\mathtt{x}$, a, b, and c are *variables* and that $\mathtt{x} \notin \{a, b, c\}$.

The small axioms for command forms are

$$\{\mathtt{x} \doteq a\}\ \ \mathtt{x} := \mathsf{E}\ \ \{\mathtt{x} \doteq \mathsf{E}[a/\mathtt{x}]\}$$

$$\{\mathsf{E} \mapsto a, b\}\ \ \mathsf{E}.\mathtt{tail} := \mathsf{E}'\ \ \{\mathsf{E} \mapsto a, \mathsf{E}')\}$$

$$\{\mathtt{x} \doteq a\}\ \ \mathtt{x} := \mathtt{new}\,\mathtt{C}(\mathsf{E}_1, \mathsf{E}_2)\ \ \{\mathtt{x} \mapsto \mathsf{E}_1[a/\mathtt{x}], \mathsf{E}_2[a/\mathtt{x}]\}$$

$$\{\mathtt{x} = a \wedge \mathsf{E} \mapsto b, c\}\ \ \mathtt{x} := \mathsf{E}.\mathtt{tail}\ \ \{\mathtt{x} = c \wedge \mathsf{E}[a/\mathtt{x}] \mapsto b, c\}$$

The small axioms state precise properties of 0- and 1-cell heaps.

The Frame *rule and other structural rules*

The small axioms gain utility when used with these structural rules; let $\mathit{modified}(S)$ be those variables that are targets of assignments in S.

$$\mathrm{Frame}: \quad \frac{\{p\}S\{p'\}}{\{p * q\}S\{p' * q\}} \quad \text{where } \mathit{modified}(S) \cap \mathit{free}(q) = \{\}$$

$$\mathrm{Consequence}: \quad \frac{p \supset p' \quad \{p'\}S\{q'\} \quad q' \supset q}{\{p\}S\{q\}}$$

$$\mathrm{Subst}: \quad \frac{\{p\}S\{q\}}{(\{p\}S\{q\})[E_1/x_1, \cdots E_k/x_k]} \quad \text{where } \{x_1, \cdots, x_k\} \supseteq \mathit{free}(p, S, q),$$

$$\text{and } \; x_i \in \mathit{modified}(S) \text{ implies } E_i \text{ is a var, } E_i \notin \mathit{free}(E_j), j \neq i$$

The Subst rule motivates the usual rule for procedure invocation (as substitution of actuals for formals).

The Frame rule embeds a result proved of a heap region into a larger heap, *justifying modular reasoning on disjoint heap regions*.

Synthesis of strongest assertions

for this example program:

```
y:= new Cell(y,x); x.tail:= y
```

we apply the small axioms to each of the two assignments:

$$\{\mathtt{y} \doteq a\}$$

$$\mathtt{y} := \mathtt{new}\ \mathtt{Cell}(\mathtt{y}, \mathtt{x})$$

$$\{\mathtt{y} \mapsto a, \mathtt{x}\}$$

$$\{\mathtt{x} \mapsto b, c\}$$

$$\mathtt{x.tail} := \mathtt{y}$$

$$\{\mathtt{x} \mapsto b, \mathtt{y}\}$$

The example: `y:= new Cell(y,x); x.tail:= y`

Next, we apply the Frame *rule to both derivations:*

$$\begin{array}{lcl}
\{y \doteq a\} & & \{y \doteq a * x \mapsto b, c\} \\
\quad y := \ \mathtt{new}\ \mathtt{Cell}(y, x) & & \quad y := \ \mathtt{new}\ \mathtt{Cell}(y, x) \\
\{y \mapsto a, x\} & & \{y \mapsto a, x * x \mapsto b, c\} \\
 & \Longrightarrow & \\
\{x \mapsto b, c\} & & \{y \mapsto a, x * x \mapsto b, c\} \\
\quad x.\mathtt{tail} := \ y & & \quad x.\mathtt{tail} := \ y \\
\{x \mapsto b, y\} & & \{y \mapsto a, x * x \mapsto b, y\}
\end{array}$$

The example: `y:= new Cell(y,x); x.tail:= y`

Now, we apply command composition:

$$
\begin{array}{lclcl}
\{y \doteq a\} & & \{y \doteq a * x \mapsto b, c\} & & \\
y := \texttt{new Cell}(y, x) & & y := \texttt{new Cell}(y, x) & & \{y \doteq a * x \mapsto b, c\} \\
\{y \mapsto a, x\} & & \{y \mapsto a, x * x \mapsto b, c\} & & \quad y := \texttt{new Cell}(y, x) \\
 & \Rightarrow & & \Rightarrow & \{y \mapsto a, x * x \mapsto b, c\} \\
\{x \mapsto b, c\} & & \{y \mapsto a, x * x \mapsto b, c\} & & \quad x.\texttt{tail} := y \\
x.\texttt{tail} := y & & x.\texttt{tail} := y & & \{y \mapsto a, x * x \mapsto b, y\} \\
\{x \mapsto b, y\} & & \{y \mapsto a, x * x \mapsto b, y\} & &
\end{array}
$$

The small axioms plus the structural rules plus the rules for the command forms are *relatively complete* for the assertion language and $\mathcal{H}eap$ models de£ned earlier.

Aliasing: `x:= new Cell(3,nil); y:= x; y.head:= 4`

$\{x \doteq a\}$

  $x := \texttt{new Cell}(3, \texttt{nil})$

$\{x \mapsto 3, nil\}$

$\{y \doteq b\}$

  $y := x$

$\{y \doteq x\}$

$\{y \mapsto c, d\}$

  $y.\mathrm{head} := 4$

$\{y \mapsto 4, d\}$

$\Longrightarrow$

$\{x \doteq a * y \doteq b\}$

  $x := \texttt{new Cell}(3, \texttt{nil})$

$\{x \mapsto 3, nil * y \doteq b\}$

  $y := x$

$\{x \mapsto 3, nil * y \doteq x\}$

$\{y \mapsto c, d\}$

  $y.\mathrm{head} := 4$

$\{y \mapsto 4, d\}$

We can try to complete the proof incorrectly

$$\{x \doteq a * y \doteq b\}$$
$$\quad x := \texttt{new Cell}(3, \texttt{nil})$$
$$\quad y := x$$
$$\{x \mapsto 3, \mathrm{nil} * y \doteq x\}$$

$$\{y \mapsto c, d\}$$
$$\quad y.\mathrm{head} := 4$$
$$\{y \mapsto 4, d\}$$

$$\{x \doteq a * y \doteq b * y \mapsto c, d\}$$
$$\quad x := \texttt{new Cell}(3, \texttt{nil})$$
$$\quad y := x$$
$$\{x \mapsto 3, \mathrm{nil} * y \doteq x * y \mapsto c, d\}$$

$$\Longrightarrow \{x \mapsto 3, \mathrm{nil} * y \doteq x * y \mapsto c, d\}$$
$$\equiv \{y \mapsto 3, \mathrm{nil} * y \mapsto c, d\}$$
$$\equiv \{\mathsf{false}\}$$
$$\quad y.\mathrm{head} := 4$$
$$\{x \mapsto 3, \mathrm{nil} * y \doteq x * y \mapsto 4, d\}$$
$$\equiv \{\mathsf{false}\}$$

The Frame rule disallows $y \mapsto c, d$ because y is modi£ed within the commands.

y cannot name two disjoint cells in the assertions.

$$\mathsf{Frame}: \quad \frac{\{p\}S\{p'\}}{\{p * q\}S\{p' * q\}} \quad \text{where } \mathit{modified}(S) \cap \mathit{free}(q) = \{\}$$

A correct proof completion uses the Frame, Subst, *and* Consequence *rules*

$$\begin{array}{l}\{x \doteq a * y \doteq b\} \\ \quad x := \texttt{new Cell}(3, \texttt{nil}) \\ \quad y := x \\ \{x \mapsto 3, nil * y \doteq x\} \\ \\ \{y \mapsto c, d\} \\ \quad y.\texttt{head} := 4 \\ \{y \mapsto 4, d\}\end{array}
\Longrightarrow
\begin{array}{l}\{x \doteq a * y \doteq b\} \\ \quad x := \texttt{new Cell}(3, \texttt{nil}) \\ \quad y := x \\ \{x \mapsto 3, nil * y \doteq x\} \\ \{y \mapsto 3, nil * y \doteq x\} \\ \\ \{y \mapsto 3, nil\} \\ \quad y.\texttt{head} := 4 \\ \{y \mapsto 4, nil\}\end{array}
\Longrightarrow
\begin{array}{l}\{x \doteq a * y \doteq b\} \\ \quad x := \texttt{new Cell}(3, \texttt{nil}) \\ \quad y := x \\ \{y \mapsto 3, nil * y \doteq x\} \\ \quad y.\texttt{head} := 4 \\ \{y \mapsto 4, nil * y \doteq x\}\end{array}$$

$$\text{Frame}: \frac{\{p\}S\{p'\}}{\{p * q\}S\{p' * q\}} \qquad \text{Consequence}: \frac{p \supset p' \quad \{p'\}S\{q'\} \quad q' \supset q}{\{p\}S\{q\}}$$

$$\text{Subst}: \frac{\{p\}S\{q\}}{(\{p\}S\{q\})[E_1/x_1, \cdots E_k/x_k]}$$

Relationship to Linear Logic

Both separation and linear logics are *substructural logics*: no weakening and no contraction:

$$A * B \not\models A \qquad\qquad A \not\models A * A$$

But separation logic's additives, $\wedge$, $\vee$, and $\rightarrow$, behave classically (or intuitionistically, if desired — partially order Heap), so that distribution and deduction hold:

$$A \wedge (B \vee C) \dashv\models (A \wedge B) \vee (A \wedge C)$$
$$A \wedge B \models C \text{ iff } A \models B \rightarrow C$$

These laws *fail* for linear logic (where $\wedge$ is $\&$, $\vee$ is $\oplus$, and $\rightarrow$ is $!(\cdot) \multimap (\cdot)$).

Separation logic has deduction for the multiplicatives:

$$A * B \models C \text{ iff } A \models B \mathrel{-\!\!*} C$$

Unlike linear logic, *there can be no* $!$ *such that* $!A \mathrel{-\!\!*} B \dashv\models A \rightarrow B$.

Practical intuition:

Use **linear logic** to study *production/consumption of short-lived resources* (e.g., to model a Petri net that produces and consumes tokens at its transitions).

Use **separation logic** to study *ownership of long-term resources* (e.g., to model a Petri net whose transitions “own” the token that arrives and “lose” ownership when the token departs — the token behaves like a “pinball” (*boule du ¤ipper*) of a pinball game).

References

This talk: `www.cis.ksu.edu/~schmidt/papers`

1. **Peter O'Hearn's web page:** `http://www.dcs.qmul.ac.uk/~ohearn/`
2. P. O'Hearn, J. Reynolds, H. Yang. Local reasoning about programs that alter data structures. `http://www.dcs.qmul.ac.uk/~ohearn/`
3. S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. Proc. 28th ACM POPL, London, 2001.
4. D. Pym, P. O'Hearn, H. Yang. Possible worlds and resources: The semantics of BI. *Theoretical Computer Science* `http://www.dcs.qmul.ac.uk/~ohearn/`
5. J. Reynolds. Separation Logic: a logic for shared mutable data structures. Proc. 17th LICS 2002.
6. M. Sagiv, T. Reps, R. Wilhelm. Parametric Shape Analysis via 3-Valued Logic. 26th ACM POPL 1999.