Permission accounting in separation logic

Richard Bornat (Mdx), Cristiano Calcagno (IC),
Peter O'Hearn (QM), Matthew Parkinson (Cam)

LRPP, Turku, 13th July 2004

Move over typing: here comes resourcing!

Move over typing: here comes resourcing!

- Resourcing is the next formal step towards program safety, following the success of typing.

Move over typing: here comes resourcing!

- Resourcing is the next formal step towards program safety, following the success of typing.
- Resourcing is about the amount of resource used by a program; typing is about the kind of resource.

Move over typing: here comes resourcing!

- Resourcing is the next formal step towards program safety, following the success of typing.
- Resourcing is about the amount of resource used by a program; typing is about the kind of resource.
- “A well-typed program won’t go wrong” (Milner).

Move over typing: here comes resourcing!

- Resourcing is the next formal step towards program safety, following the success of typing.
- Resourcing is about the amount of resource used by a program; typing is about the kind of resource.
- “A well-typed program won’t go wrong” (Milner).
- “Well-resourced programs mind their own business” (O’Hearn).

A program in need of resourcing

$\mathrm{P}(\mathit{read});$
$\quad \text{if } \mathit{count} = 0 \text{ then } \mathrm{P}(\mathit{write})$
$\qquad\qquad \text{else skip fi};$
$\quad \mathit{count}{+} := 1;$
$\mathrm{V}(\mathit{read});$

... reading happens here ...

$\mathrm{P}(\mathit{read});$
$\quad \mathit{count}{-} := 1;$
$\quad \text{if } \mathit{count} = 0 \text{ then } \mathrm{V}(\mathit{write})$
$\qquad\qquad \text{else skip fi};$
$\mathrm{V}(\mathit{read})$

$\mathrm{P}(\mathit{write});$

... writing happens here ...

$\mathrm{V}(\mathit{write})$

Separation logic

Separation logic

- Just a bastard child of BI (Pym, O’Hearn).

Separation logic

- Just a bastard child of BI (Pym, O'Hearn).
- $E \mapsto E'$ (points to) is permission to read/write/dispose cell at heap address E with contents E'.

Separation logic

- Just a bastard child of BI (Pym, O'Hearn).
- $E \mapsto E'$ (points to) is permission to read/write/dispose cell at heap address E with contents E'.
- Previously $\mapsto$ was ownership; before that a heap predicate (and it still is).

Separation logic

- Just a bastard child of BI (Pym, O'Hearn).
- $E \mapsto E'$ (points to) is permission to read/write/dispose cell at heap address E with contents E'.
- Previously $\mapsto$ was ownership; before that a heap predicate (and it still is).
- $\mathbf{emp}$ is no permission.

Separation logic

- Just a bastard child of BI (Pym, O'Hearn).
- $E \mapsto E'$ (points to) is permission to read/write/dispose cell at heap address E with contents E'.
- Previously $\mapsto$ was ownership; before that a heap predicate (and it still is).
- $\mathbf{emp}$ is no permission.
- $A \star B$ (star) is separation of resource.

Separation logic

- Just a bastard child of BI (Pym, O'Hearn).
- $E \mapsto E'$ (points to) is permission to read/write/dispose cell at heap address E with contents E'.
- Previously $\mapsto$ was ownership; before that a heap predicate (and it still is).
- $\mathbf{emp}$ is no permission.
- $A \star B$ (star) is separation of resource.
- $A \wedge B$ (and) is identity of resource.

Separation logic

- Just a bastard child of BI (Pym, O'Hearn).
- $E \mapsto E'$ (points to) is permission to read/write/dispose cell at heap address E with contents E'.
- Previously $\mapsto$ was ownership; before that a heap predicate (and it still is).
- $\mathbf{emp}$ is no permission.
- $A \star B$ (star) is separation of resource.
- $A \wedge B$ (and) is identity of resource.
- $A \wedge (B \star \mathrm{true})$ is all A, partly B.

Framing, hence small axioms

Framing, hence small axioms

$$\frac{\{Q\}C\{R\}}{\{P \star Q\}C\{P \star R\}} \quad (\mathit{modifies}\ \boldsymbol{C} \cap \mathit{vars}\ \boldsymbol{P} = \emptyset)$$

Framing, hence small axioms

$$\frac{\{Q\}C\{R\}}{\{P \star Q\}C\{P \star R\}} \quad (\mathit{modifies}\ C \cap \mathit{vars}\ P = \emptyset)$$

$$
\begin{array}{rll}
\{R^x_E\} & x{:=}E & \{R\} \\
\{x \mapsto \_\} & [x]{:=}E & \{x \mapsto E\} \\
\{E' \mapsto E\} & x{:=}[E'] & \{E' \mapsto E \wedge x = E\} \ (x \text{ not free in } E, E') \\
\{\mathbf{emp}\} & x{:=}\mathrm{new}(E) & \{x \mapsto E\} \\
\{E \mapsto \_\} & \mathrm{dispose}\, E & \{\mathbf{emp}\}
\end{array}
$$

Concurrency rules

Concurrency rules

$$\frac{\{Q_1\}\,C_1\,\{R_1\}\ \cdots\ \{Q_n\}\,C_n\,\{R_n\}}{\{Q_1 \star \cdots \star Q_n\}\quad (C_1 \parallel \cdots \parallel C_n)\{R_1 \star \cdots \star R_n\}}\ \text{(non-interference-of-variables)}$$

Concurrency rules

$$\frac{\{Q_1\}\, C_1\, \{R_1\} \cdots \{Q_n\}\, C_n\, \{R_n\}}{\{Q_1 \star \cdots \star Q_n\} \quad (C_1 \parallel \cdots \parallel C_n)\{R_1 \star \cdots \star R_n\}} \text{ (non-interference-of-variables)}$$

$$\frac{\{(Q \star I_r) \wedge B\}C\{R \star I_r\}}{\{Q\}\text{with } r \text{ when } B \text{ do } C \text{ od}\{R\}} \text{ (non-interference-of-variables)}$$

Concurrency rules

$$\frac{\{Q_1\}\ C_1\ \{R_1\}\ \cdots\ \{Q_n\}\ C_n\ \{R_n\}}{\{Q_1 \star \cdots \star Q_n\}\quad (C_1 \parallel \cdots \parallel C_n)\{R_1 \star \cdots \star R_n\}}\ \text{(non-interference-of-variables)}$$

$$\frac{\{(Q \star I_r) \wedge B\}C\{R \star I_r\}}{\{Q\}\mathrm{with}\ r\ \mathrm{when}\ B\ \mathrm{do}\ C\ \mathrm{od}\{R\}}\ \text{(non-interference-of-variables)}$$

- Both proved sound by Brookes.

Concurrency rules

$$\frac{\{Q_1\}\, C_1\, \{R_1\} \cdots \{Q_n\}\, C_n\, \{R_n\}}{\{Q_1 \star \cdots \star Q_n\} \quad (C_1 \parallel \cdots \parallel C_n)\{R_1 \star \cdots \star R_n\}} \text{ (non-interference-of-variables)}$$

$$\frac{\{(Q \star I_r) \wedge B\} C \{R \star I_r\}}{\{Q\}\mathrm{with}\ r\ \mathrm{when}\ B\ \mathrm{do}\ C\ \mathrm{od}\{R\}} \text{ (non-interference-of-variables)}$$

- Both proved sound by Brookes.
- A version of the CCR rule covers semaphores, in which C is either $m := m + 1$ or $m := m - 1$.

The ownership trick (O'Hearn)

Resource r : Vars full, b;

$$
\left(
\begin{array}{l}
x := \mathrm{new}(); \\
\text{with } r \text{ when } \neg \mathit{full} \text{ do} \\
\quad b := x; \\
\quad \mathit{full} := \text{true} \\
\text{od}
\end{array}
\;\middle\|\;
\begin{array}{l}
\text{with } r \text{ when } \mathit{full} \text{ do} \\
\quad y := b; \\
\quad \mathit{full} := \text{false} \\
\text{od}; \\
\text{dispose } y
\end{array}
\right)
$$

The ownership trick (O'Hearn)

Resource r : Vars full, b;

$$
\left(
\begin{array}{l}
\{\mathbf{emp}\} \\
x := \mathrm{new}(); \\
\mathrm{with}\ r\ \mathrm{when}\ \neg \mathit{full}\ \mathrm{do} \\
\quad b := x; \\
\quad \mathit{full} := \mathrm{true} \\
\mathrm{od} \\
\{\mathbf{emp}\}
\end{array}
\right\|
\begin{array}{l}
\{\mathbf{emp}\} \\
\mathrm{with}\ r\ \mathrm{when}\ \mathit{full}\ \mathrm{do} \\
\quad y := b; \\
\quad \mathit{full} := \mathrm{false} \\
\mathrm{od}; \\
\mathrm{dispose}\, y \\
\{\mathbf{emp}\}
\end{array}
\right)
$$

The ownership trick (O'Hearn)

Resource r : Vars full, b;
Invariant $(\mathit{full} \wedge b \mapsto \_) \vee (\neg \mathit{full} \wedge \mathbf{emp})$

$$
\left(
\begin{array}{l}
\{\mathbf{emp}\}\\
x := \mathrm{new}();\\
\text{with } r \text{ when } \neg \mathit{full} \text{ do}\\
\quad b := x;\\
\quad \mathit{full} := \text{true}\\
\text{od}\\
\{\mathbf{emp}\}
\end{array}
\right\|
\begin{array}{l}
\{\mathbf{emp}\}\\
\text{with } r \text{ when } \mathit{full} \text{ do}\\
\quad y := b;\\
\quad \mathit{full} := \text{false}\\
\text{od};\\
\text{dispose } y\\
\{\mathbf{emp}\}
\end{array}
\right)
$$

The ownership trick (O'Hearn)

$$
\begin{array}{l}
\text{Resource } r : \text{Vars } \mathit{full}, b; \\
\qquad\qquad \text{Invariant } (\mathit{full} \wedge b \mapsto \_) \vee (\neg \mathit{full} \wedge \mathbf{emp})
\end{array}
$$

$$
\left(
\begin{array}{l}
\{\mathbf{emp}\} \\
x := \mathrm{new}(); \\
\{x \mapsto \_\} \\
\text{with } r \text{ when } \neg \mathit{full} \text{ do} \\
\quad b := x; \\
\quad \mathit{full} := \text{true} \\
\text{od} \\
\{\mathbf{emp}\}
\end{array}
\;\middle\|\;
\begin{array}{l}
\{\mathbf{emp}\} \\
\text{with } r \text{ when } \mathit{full} \text{ do} \\
\quad y := b; \\
\quad \mathit{full} := \text{false} \\
\text{od;} \\
\text{dispose } y \\
\{\mathbf{emp}\}
\end{array}
\right)
$$

The ownership trick (O'Hearn)

$$
\begin{array}{ll}
\text{Resource } r : & \text{Vars } \mathit{full}, b; \\
& \text{Invariant } (\mathit{full} \wedge b \mapsto \_) \vee (\neg \mathit{full} \wedge \mathbf{emp})
\end{array}
$$

$$
\left(
\begin{array}{l||l}
\{\mathbf{emp}\} & \{\mathbf{emp}\} \\
x := \text{new}(); & \text{with } r \text{ when } \mathit{full} \text{ do} \\
\{x \mapsto \_\} & \\
\text{with } r \text{ when } \neg \mathit{full} \text{ do} & \quad y := b; \\
\quad \{\neg \mathit{full} \wedge \mathbf{emp} \star x \mapsto \_\} & \\
\quad b := x; & \quad \mathit{full} := \text{false} \\
& \\
\quad \mathit{full} := \text{true} & \text{od}; \\
& \\
\text{od} & \text{dispose } y \\
\{\mathbf{emp}\} & \{\mathbf{emp}\}
\end{array}
\right)
$$

The ownership trick (O'Hearn)

$$
\begin{array}{ll}
\text{Resource } r : & \text{Vars } \mathit{full}, b; \\
& \text{Invariant } (\mathit{full} \wedge b \mapsto \_) \vee (\neg\mathit{full} \wedge \mathbf{emp})
\end{array}
$$

$$
\left(
\begin{array}{l|l}
\{\mathbf{emp}\} & \{\mathbf{emp}\} \\
x := \mathrm{new}(); & \text{with } r \text{ when } \mathit{full} \text{ do} \\
\{x \mapsto \_\} & \\
\text{with } r \text{ when } \neg\mathit{full} \text{ do} & \quad y := b; \\
\quad \{\neg\mathit{full} \wedge \mathbf{emp} \star x \mapsto \_\} & \\
\quad b := x; & \quad \mathit{full} := \text{false} \\
\quad \{\neg\mathit{full} \wedge \mathbf{emp} \star x \mapsto \_ \wedge b = x\} & \\
\quad \mathit{full} := \text{true} & \text{od}; \\
& \\
\text{od} & \text{dispose}\, y \\
\{\mathbf{emp}\} & \{\mathbf{emp}\}
\end{array}
\right)
$$

The ownership trick (O'Hearn)

$$
\begin{array}{l}
\text{Resource } r : \text{Vars } \mathit{full}, b; \\
\qquad\qquad \text{Invariant } (\mathit{full} \wedge b \mapsto \_) \vee (\neg \mathit{full} \wedge \mathbf{emp})
\end{array}
$$

$$
\left(
\begin{array}{l||l}
\{\mathbf{emp}\} & \{\mathbf{emp}\} \\
x := \mathrm{new}(); & \text{with } r \text{ when } \mathit{full} \text{ do} \\
\{x \mapsto \_\} & \\
\text{with } r \text{ when } \neg \mathit{full} \text{ do} & \quad y := b; \\
\quad \{\neg \mathit{full} \wedge \mathbf{emp} \star x \mapsto \_\} & \\
\quad b := x; & \quad \mathit{full} := \mathrm{false} \\
\quad \{\neg \mathit{full} \wedge \mathbf{emp} \star x \mapsto \_ \wedge b = x\} & \\
\quad \mathit{full} := \mathrm{true} & \text{od}; \\
\quad \{\mathit{full} \wedge b \mapsto \_ \star \mathbf{emp}\} & \\
\text{od} & \text{dispose } y \\
\{\mathbf{emp}\} & \{\mathbf{emp}\}
\end{array}
\right)
$$

The ownership trick (O'Hearn)

$$
\begin{array}{l}
\text{Resource } r : \text{Vars } \mathit{full}, b; \\
\qquad\qquad\qquad \text{Invariant } (\mathit{full} \wedge b \mapsto \_) \vee (\neg\mathit{full} \wedge \mathbf{emp})
\end{array}
$$

$$
\left(
\begin{array}{l}
\{\mathbf{emp}\} \\
x := \text{new}(); \\
\{x \mapsto \_\} \\
\text{with } r \text{ when } \neg\mathit{full} \text{ do} \\
\quad \{\neg\mathit{full} \wedge \mathbf{emp} \star x \mapsto \_\} \\
\quad b := x; \\
\quad \{\neg\mathit{full} \wedge \mathbf{emp} \star x \mapsto \_ \wedge b = x\} \\
\quad \mathit{full} := \text{true} \\
\quad \{\mathit{full} \wedge b \mapsto \_ \star \mathbf{emp}\} \\
\text{od} \\
\{\mathbf{emp}\}
\end{array}
\;\Bigg\|\;
\begin{array}{l}
\{\mathbf{emp}\} \\
\text{with } r \text{ when } \mathit{full} \text{ do} \\
\quad \{\mathit{full} \wedge b \mapsto \_ \star \mathbf{emp}\} \\
\quad y := b; \\
\\
\quad \mathit{full} := \text{false} \\
\\
\text{od}; \\
\\
\text{dispose } y \\
\{\mathbf{emp}\}
\end{array}
\right)
$$

The ownership trick (O'Hearn)

$$\begin{array}{l} \text{Resource } r : \text{Vars } \mathit{full}, b; \\ \qquad\qquad \text{Invariant } (\mathit{full} \wedge b \mapsto \_) \vee (\neg\mathit{full} \wedge \mathbf{emp}) \end{array}$$

$$\left(\begin{array}{l|l}
\{\mathbf{emp}\} & \{\mathbf{emp}\} \\
x := \text{new}(); & \text{with } r \text{ when } \mathit{full} \text{ do} \\
\{x \mapsto _\} & \quad \{\mathit{full} \wedge b \mapsto _ \star \mathbf{emp}\} \\
\text{with } r \text{ when } \neg\mathit{full} \text{ do} & \quad y := b; \\
\quad \{\neg\mathit{full} \wedge \mathbf{emp} \star x \mapsto _\} & \quad \{\mathit{full} \wedge b \mapsto _ \star \mathbf{emp} \wedge y = b\} \\
\quad b := x; & \quad \mathit{full} := \text{false} \\
\quad \{\neg\mathit{full} \wedge \mathbf{emp} \star x \mapsto _ \wedge b = x\} & \\
\quad \mathit{full} := \text{true} & \text{od}; \\
\quad \{\mathit{full} \wedge b \mapsto _ \star \mathbf{emp}\} & \\
\text{od} & \text{dispose } y \\
\{\mathbf{emp}\} & \{\mathbf{emp}\}
\end{array}\right)$$

The ownership trick (O'Hearn)

$$\begin{array}{l} \text{Resource } r : \text{Vars } \mathit{full}, b; \\ \qquad\qquad \text{Invariant } (\mathit{full} \wedge b \mapsto \_) \vee (\neg \mathit{full} \wedge \mathbf{emp}) \end{array}$$

$$\left(\begin{array}{l} \{\mathbf{emp}\} \\ x := \text{new}(); \\ \{x \mapsto \_\} \\ \text{with } r \text{ when } \neg \mathit{full} \text{ do} \\ \quad \{\neg \mathit{full} \wedge \mathbf{emp} \star x \mapsto \_\} \\ \quad b := x; \\ \quad \{\neg \mathit{full} \wedge \mathbf{emp} \star x \mapsto \_ \wedge b = x\} \\ \quad \mathit{full} := \text{true} \\ \quad \{\mathit{full} \wedge b \mapsto \_ \star \mathbf{emp}\} \\ \text{od} \\ \{\mathbf{emp}\} \end{array} \middle\| \begin{array}{l} \{\mathbf{emp}\} \\ \text{with } r \text{ when } \mathit{full} \text{ do} \\ \quad \{\mathit{full} \wedge b \mapsto \_ \star \mathbf{emp}\} \\ \quad y := b; \\ \quad \{\mathit{full} \wedge b \mapsto \_ \star \mathbf{emp} \wedge y = b\} \\ \quad \mathit{full} := \text{false} \\ \quad \{\neg \mathit{full} \wedge \mathbf{emp} \star y \mapsto \_\} \\ \text{od}; \\ \\ \text{dispose } y \\ \{\mathbf{emp}\} \end{array} \right)$$

The ownership trick (O'Hearn)

Resource r : Vars full, b;
Invariant $(\mathit{full} \wedge b \mapsto \_) \vee (\neg \mathit{full} \wedge \mathbf{emp})$

$$
\left(
\begin{array}{l}
\{\mathbf{emp}\} \\
x := \mathrm{new}(); \\
\{x \mapsto \_\} \\
\text{with } r \text{ when } \neg \mathit{full} \text{ do} \\
\quad \{\neg \mathit{full} \wedge \mathbf{emp} \star x \mapsto \_\} \\
\quad b := x; \\
\quad \{\neg \mathit{full} \wedge \mathbf{emp} \star x \mapsto \_ \wedge b = x\} \\
\quad \mathit{full} := \text{true} \\
\quad \{\mathit{full} \wedge b \mapsto \_ \star \mathbf{emp}\} \\
\text{od} \\
\{\mathbf{emp}\}
\end{array}
\;\middle\|\;
\begin{array}{l}
\{\mathbf{emp}\} \\
\text{with } r \text{ when } \mathit{full} \text{ do} \\
\quad \{\mathit{full} \wedge b \mapsto \_ \star \mathbf{emp}\} \\
\quad y := b; \\
\quad \{\mathit{full} \wedge b \mapsto \_ \star \mathbf{emp} \wedge y = b\} \\
\quad \mathit{full} := \text{false} \\
\quad \{\neg \mathit{full} \wedge \mathbf{emp} \star y \mapsto \_\} \\
\text{od}; \\
\{y \mapsto \_\} \\
\text{dispose } y \\
\{\mathbf{emp}\}
\end{array}
\right)
$$

Part I

Passivity, sharing, accounting

Passivity

Passivity

- Passivity is a property of a program and a resource: the program doesn't change the contents of the resource.

Passivity

- Passivity is a property of a program and a resource: the program doesn't change the contents of the resource.
- We want to specify passivity by specifying a read-only resource.

Passivity

- Passivity is a property of a program and a resource: the program doesn't change the contents of the resource.
- We want to specify passivity by specifying a read-only resource.
- We require that a program, given a read-only resource, cannot change its contents.

Splitting and sharing

Splitting and sharing

- Since Dijkstra, we have known that we can safely share read-only resources.

Splitting and sharing

- Since Dijkstra, we have known that we can safely share read-only resources.
- Total permission $E \mapsto E'$, given by new, allows read/write/dispose.

Splitting and sharing

- Since Dijkstra, we have known that we can safely share read-only resources.
- Total permission $E \mapsto E'$, given by new, allows read/write/dispose.
- Concurrent read permissions must be $(\star)$ separable, because of the concurrency rule.

Accounting

Accounting

- Splitting into multiple read permissions is easy.

Accounting

- Splitting into multiple read permissions is easy.
- To write or dispose we have to know when we have all the read permissions back.

Accounting

- Splitting into multiple read permissions is easy.
- To write or dispose we have to know when we have all the read permissions back.
- A program which doesn't keep account leaks resource.

Boyland's suggestion: $\frac{1}{2} + \frac{1}{2} = 1$

Boyland's suggestion: $\frac{1}{2} + \frac{1}{2} = 1$

- Boyland (Wisconsin) developed a means of permission accounting in disjoint concurrency, dealing with variables and heap locations.

Boyland's suggestion: $\frac{1}{2} + \frac{1}{2} = 1$

- Boyland (Wisconsin) developed a means of permission accounting in disjoint concurrency, dealing with variables and heap locations.
- He associates a number z with each permission: $z = 1$ total; $0 < z < 1$ read-only.

Boyland's suggestion: $\frac{1}{2} + \frac{1}{2} = 1$

- Boyland (Wisconsin) developed a means of permission accounting in disjoint concurrency, dealing with variables and heap locations.
- He associates a number z with each permission: $z = 1$ total; $0 < z < 1$ read-only.
- Fractional permissions are specification-only (cf. types).

Boyland's suggestion: $\frac{1}{2} + \frac{1}{2} = 1$

- Boyland (Wisconsin) developed a means of permission accounting in disjoint concurrency, dealing with variables and heap locations.
- He associates a number z with each permission: $z = 1$ total; $0 < z < 1$ read-only.
- Fractional permissions are specification-only (cf. types).
- In practice the arithmetic is very easy: fractions are simpler to use than (e.g.) sets of binary trees.

Boyland's suggestion: $\frac{1}{2} + \frac{1}{2} = 1$

- Boyland (Wisconsin) developed a means of permission accounting in disjoint concurrency, dealing with variables and heap locations.
- He associates a number z with each permission: $z = 1$ total; $0 < z < 1$ read-only.
- Fractional permissions are specification-only (cf. types).
- In practice the arithmetic is very easy: fractions are simpler to use than (e.g.) sets of binary trees.
- The magnitude of non-integral fractions doesn't matter, except as a matter of accounting.

A fractional model

A fractional model

- Heaps are now partial maps from Nat to (int, fraction). (Previously Nat to int.)

A fractional model

- Heaps are now partial maps from Nat to (int, fraction). (Previously Nat to int.)
- A simpler model – just read / total permissions – fails to account and doesn't have the frame property.

Proof theory

Proof theory

$$
\begin{aligned}
E \xmapsto[z]{} E' \quad &\Rightarrow \quad 0 < z \leq 1 \\
E \xmapsto[z+z']{} E' \wedge z > 0 \wedge z' > 0 \ &\Longleftrightarrow \ E \xmapsto[z]{} E' \star E \xmapsto[z']{} E'
\end{aligned}
$$

Proof theory

$$E \underset{z}{\mapsto} E' \quad \Rightarrow \quad 0 < z \leq 1$$

$$E \underset{z+z'}{\longmapsto} E' \wedge z > 0 \wedge z' > 0 \iff E \underset{z}{\mapsto} E' \star E \underset{z'}{\mapsto} E'$$

$$\begin{array}{rcl}
\left\{R^x_E\right\} & x{:=}E & \left\{R\right\} \\
\left\{E' \underset{1}{\mapsto} _\right\} & [E']{:=}E & \left\{E' \underset{1}{\mapsto} E\right\} \\
\left\{E' \underset{z}{\mapsto} E\right\} & x{:=}[E'] & \left\{E' \underset{z}{\mapsto} E \wedge x = E\right\} \; (x \text{ not free in } E, E') \\
\left\{\mathbf{emp}\right\} & x{:=}\mathrm{new}(E) & \left\{x \underset{1}{\mapsto} E\right\} \\
\left\{E \underset{1}{\mapsto} _\right\} & \mathrm{dispose}\, E & \left\{\mathbf{emp}\right\}
\end{array}$$

Proof theory

$$E \underset{z}{\mapsto} E' \quad \Rightarrow \quad 0 < z \leq 1$$

$$E \underset{z+z'}{\longmapsto} E' \wedge z > 0 \wedge z' > 0 \iff E \underset{z}{\mapsto} E' \star E \underset{z'}{\mapsto} E'$$

$$\begin{array}{rll}
\left\{R^x_E\right\} & x{:=}E & \left\{R\right\} \\
\left\{E' \underset{1}{\mapsto} _\right\} & [E']{:=}E & \left\{E' \underset{1}{\mapsto} E\right\} \\
\left\{E' \underset{z}{\mapsto} E\right\} & x{:=}[E'] & \left\{E' \underset{z}{\mapsto} E \wedge x = E\right\} \ (x \text{ not free in } E, E') \\
\left\{\mathbf{emp}\right\} & x{:=}\mathrm{new}(E) & \left\{x \underset{1}{\mapsto} E\right\} \\
\left\{E \underset{1}{\mapsto} _\right\} & \mathrm{dispose}\, E & \left\{\mathbf{emp}\right\}
\end{array}$$

- Not (yet) proved sound by Brookes. (But surely ...)

Proof

$\{\mathbf{emp}\}$
$x := \mathrm{new}();$

$[x] := 1;$

$$\left(y := [x] \quad \Big\| \quad z := [x] + 1 \right);$$

$\mathrm{dispose}\, x$
$\{\mathbf{emp} \wedge y = 1 \wedge z = 2\}$

Proof

$\{\mathbf{emp}\}$
$x := \mathrm{new}();$
$\left\{x \underset{1}{\longmapsto} \_\right\}$
$[x] := 1;$

$$\left(y := [x] \quad \Big\| \quad z := [x] + 1 \right);$$

$\mathrm{dispose}\, x$
$\{\mathbf{emp} \wedge y = 1 \wedge z = 2\}$

Proof

$$\{\mathbf{emp}\}$$
$$x := \mathrm{new}();$$
$$\left\{x \underset{1}{\mapsto} \_\right\}$$
$$[x] := 1;$$
$$\left\{x \underset{1}{\mapsto} 1\right\}$$
$$\left(y := [x] \quad \Bigg\| \quad z := [x] + 1 \right);$$

$$\mathrm{dispose}\, x$$
$$\{\mathbf{emp} \wedge y = 1 \wedge z = 2\}$$

Proof

$\{\mathbf{emp}\}$
$x := \mathrm{new}();$
$\left\{x \underset{1}{\longmapsto} \_\right\}$
$[x] := 1;$
$\left\{x \underset{1}{\longmapsto} 1\right\} \therefore \left\{x \underset{0.5}{\longmapsto} 1 \star x \underset{0.5}{\longmapsto} 1\right\}$

$$\left(y := [x] \quad \Bigg\| \quad z := [x] + 1 \right);$$

dispose x
$\{\mathbf{emp} \wedge y = 1 \wedge z = 2\}$

Proof

$$\{\mathbf{emp}\}$$
$$x := \text{new}();$$
$$\left\{x \underset{1}{\longmapsto} \_\right\}$$
$$[x] := 1;$$
$$\left\{x \underset{1}{\longmapsto} 1\right\} \therefore \left\{x \underset{0.5}{\longmapsto} 1 \star x \underset{0.5}{\longmapsto} 1\right\}$$
$$\left(\begin{array}{l|l} \left\{x \underset{0.5}{\longmapsto} 1\right\} & \left\{x \underset{0.5}{\longmapsto} 1\right\} \\ y := [x] & z := [x] + 1 \\ & \end{array}\right);$$

$$\text{dispose}\, x$$
$$\{\mathbf{emp} \wedge y = 1 \wedge z = 2\}$$

Proof

$$
\begin{array}{l}
\{\mathbf{emp}\} \\
x := \mathrm{new}(); \\
\left\{x \underset{1}{\longmapsto} \_\right\} \\
[x] := 1; \\
\left\{x \underset{1}{\longmapsto} 1\right\} \therefore \left\{x \underset{0.5}{\longmapsto} 1 \star x \underset{0.5}{\longmapsto} 1\right\} \\
\left(\begin{array}{l|l}
\left\{x \underset{0.5}{\longmapsto} 1\right\} & \left\{x \underset{0.5}{\longmapsto} 1\right\} \\
y := [x] & z := [x] + 1 \\
\left\{x \underset{0.5}{\longmapsto} 1 \wedge y = 1\right\} &
\end{array}\right); \\
\\
\mathrm{dispose}\, x \\
\{\mathbf{emp} \wedge y = 1 \wedge z = 2\}
\end{array}
$$

Proof

$$\{\mathbf{emp}\}$$
$$x := \mathrm{new}();$$
$$\left\{x \underset{1}{\longmapsto} \_\right\}$$
$$[x] := 1;$$
$$\left\{x \underset{1}{\longmapsto} 1\right\} \therefore \left\{x \underset{0.5}{\longmapsto} 1 \star x \underset{0.5}{\longmapsto} 1\right\}$$
$$\left(\begin{array}{l|l} \left\{x \underset{0.5}{\longmapsto} 1\right\} & \left\{x \underset{0.5}{\longmapsto} 1\right\} \\ y := [x] & z := [x] + 1 \\ \left\{x \underset{0.5}{\longmapsto} 1 \wedge y = 1\right\} & \left\{x \underset{0.5}{\longmapsto} 1 \wedge z = 2\right\} \end{array}\right);$$

$$\mathrm{dispose}\, x$$
$$\{\mathbf{emp} \wedge y = 1 \wedge z = 2\}$$

Proof

$$\{\mathbf{emp}\}$$

$$x := \mathrm{new}();$$

$$\left\{x \xmapsto[1]{} \_\right\}$$

$$[x] := 1;$$

$$\left\{x \xmapsto[1]{} 1\right\} \therefore \left\{x \xmapsto[0.5]{} 1 \star x \xmapsto[0.5]{} 1\right\}$$

$$\left(\begin{array}{l|l} \left\{x \xmapsto[0.5]{} 1\right\} & \left\{x \xmapsto[0.5]{} 1\right\} \\ y := [x] & z := [x] + 1 \\ \left\{x \xmapsto[0.5]{} 1 \wedge y = 1\right\} & \left\{x \xmapsto[0.5]{} 1 \wedge z = 2\right\} \end{array}\right);$$

$$\left\{(x \xmapsto[0.5]{} 1 \wedge y = 1) \star (x \xmapsto[0.5]{} 1 \wedge z = 2)\right\}$$

$$\mathrm{dispose}\, x$$

$$\{\mathbf{emp} \wedge y = 1 \wedge z = 2\}$$

Proof

$$
\begin{array}{l}
\{\mathbf{emp}\} \\
x := \mathrm{new}(); \\
\left\{x \xmapsto[1]{} \_\right\} \\
[x] := 1; \\
\left\{x \xmapsto[1]{} 1\right\} \therefore \left\{x \xmapsto[0.5]{} 1 \star x \xmapsto[0.5]{} 1\right\} \\
\left(\begin{array}{l|l}
\left\{x \xmapsto[0.5]{} 1\right\} & \left\{x \xmapsto[0.5]{} 1\right\} \\
y := [x] & z := [x] + 1 \\
\left\{x \xmapsto[0.5]{} 1 \wedge y = 1\right\} & \left\{x \xmapsto[0.5]{} 1 \wedge z = 2\right\}
\end{array}\right); \\
\left\{(x \xmapsto[0.5]{} 1 \wedge y = 1) \star (x \xmapsto[0.5]{} 1 \wedge z = 2)\right\} \therefore \left\{x \xmapsto[1]{} 1 \wedge y = 1 \wedge z = 2\right\} \\
\mathrm{dispose}\, x \\
\{\mathbf{emp} \wedge y = 1 \wedge z = 2\}
\end{array}
$$

Proof

$$\{\mathbf{emp}\}$$

$$x := \mathrm{new}();$$

$$\left\{x \xmapsto[1]{} \_\right\}$$

$$[x] := 1;$$

$$\left\{x \xmapsto[1]{} 1\right\} \therefore \left\{x \xmapsto[0.5]{} 1 \star x \xmapsto[0.5]{} 1\right\}$$

$$\left(\begin{array}{l|l} \left\{x \xmapsto[0.5]{} 1\right\} & \left\{x \xmapsto[0.5]{} 1\right\} \\ y := [x] & z := [x] + 1 \\ \left\{x \xmapsto[0.5]{} 1 \wedge y = 1\right\} & \left\{x \xmapsto[0.5]{} 1 \wedge z = 2\right\} \end{array}\right);$$

$$\left\{(x \xmapsto[0.5]{} 1 \wedge y = 1) \star (x \xmapsto[0.5]{} 1 \wedge z = 2)\right\} \therefore \left\{x \xmapsto[1]{} 1 \wedge y = 1 \wedge z = 2\right\}$$

$$\mathrm{dispose}\, x$$

$$\{\mathbf{emp} \wedge y = 1 \wedge z = 2\}$$

- That is exactly how hard it is to use fractional permissions.

UnProof

$$\{\mathbf{emp}\}$$

$$x := \mathrm{new}();$$

$$\left\{x \xmapsto[1]{} \_\right\}$$

$$[x] := 1;$$

$$\left\{x \xmapsto[1]{} 1\right\} \therefore \left\{x \xmapsto[0.5]{} 1 \star x \xmapsto[0.5]{} 1\right\}$$

$$\left(\begin{array}{l||l}
\left\{x \xmapsto[0.5]{} 1\right\} & \left\{x \xmapsto[0.5]{} 1\right\} \\
y := [x]; & [x] := 2; \\
 & \\
\mathrm{dispose}\, x & z := [x] + 1
\end{array}\right)$$

$$[x] := y + z$$

UnProof

$$\{\mathbf{emp}\}$$
$$x := \mathrm{new}();$$
$$\left\{x \underset{1}{\longmapsto} \_\right\}$$
$$[x] := 1;$$
$$\left\{x \underset{1}{\longmapsto} 1\right\} \therefore \left\{x \underset{0.5}{\longmapsto} 1 \star x \underset{0.5}{\longmapsto} 1\right\}$$
$$\left(\begin{array}{l|l}
\left\{x \underset{0.5}{\longmapsto} 1\right\} & \left\{x \underset{0.5}{\longmapsto} 1\right\} \\
y := [x]; & [x] := 2; \\
\left\{x \underset{0.5}{\longmapsto} 1 \wedge y = 1\right\} & \\
\mathrm{dispose}\, x & z := [x] + 1
\end{array}\right)$$

$$[x] := y + z$$

UnProof

$$\{\mathbf{emp}\}$$

$$x := \mathrm{new}();$$

$$\left\{x \underset{1}{\longmapsto} \_\right\}$$

$$[x] := 1;$$

$$\left\{x \underset{1}{\longmapsto} 1\right\} \therefore \left\{x \underset{0.5}{\longmapsto} 1 \star x \underset{0.5}{\longmapsto} 1\right\}$$

$$\left(\begin{array}{l||l}
\left\{x \underset{0.5}{\longmapsto} 1\right\} & \left\{x \underset{0.5}{\longmapsto} 1\right\} \\
y := [x]; & [x] := 2; \\
\left\{x \underset{0.5}{\longmapsto} 1 \wedge y = 1\right\} & \\
\mathrm{dispose}\, x & z := [x] + 1 \\
\{??\} &
\end{array}\right)$$

$$[x] := y + z$$

UnProof

$$\{\mathbf{emp}\}$$
$$x := \mathrm{new}();$$
$$\left\{x \underset{1}{\longmapsto} \_\right\}$$
$$[x] := 1;$$
$$\left\{x \underset{1}{\longmapsto} 1\right\} \therefore \left\{x \underset{0.5}{\longmapsto} 1 \star x \underset{0.5}{\longmapsto} 1\right\}$$
$$\left(\begin{array}{l|l} \left\{x \underset{0.5}{\longmapsto} 1\right\} & \left\{x \underset{0.5}{\longmapsto} 1\right\} \\ y := [x]; & [x] := 2; \\ \left\{x \underset{0.5}{\longmapsto} 1 \wedge y = 1\right\} & \{??\} \\ \mathrm{dispose}\, x & z := [x] + 1 \\ \{??\} & \end{array}\right)$$

$$[x] := y + z$$

UnProof

$$\{\mathbf{emp}\}$$
$$x := \mathrm{new}();$$
$$\left\{x \underset{1}{\longmapsto} \_\right\}$$
$$[x] := 1;$$
$$\left\{x \underset{1}{\longmapsto} 1\right\} \therefore \left\{x \underset{0.5}{\longmapsto} 1 \star x \underset{0.5}{\longmapsto} 1\right\}$$
$$\left(\begin{array}{l||l}
\left\{x \underset{0.5}{\longmapsto} 1\right\} & \left\{x \underset{0.5}{\longmapsto} 1\right\} \\
y := [x]; & [x] := 2; \\
\left\{x \underset{0.5}{\longmapsto} 1 \wedge y = 1\right\} & \{??\} \\
\mathrm{dispose}\, x & z := [x] + 1 \\
\{??\} & \{??\}
\end{array}\right)$$
$$\{??\}$$
$$[x] := y + z$$

Passivity and fractions

Termination Monotonicity: if C must terminate normally in h and $h \star h'$ is defined, then C must terminate normally in $h \star h'$.

Passivity and fractions

Termination Monotonicity: if C must terminate normally in h and $h \star h'$ is defined, then C must terminate normally in $h \star h'$.

- We can prove termination monotonicity for all commands in our language.

Passivity and fractions

Termination Monotonicity: if C must terminate normally in h and $h \star h'$ is defined, then C must terminate normally in $h \star h'$.

- We can prove termination monotonicity for all commands in our language.
- Suppose $\left\{10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N+1\right\}$, and it terminates.

Passivity and fractions

Termination Monotonicity: if C must terminate normally in h and $h \star h'$ is defined, then C must terminate normally in $h \star h'$.

- We can prove termination monotonicity for all commands in our language.
- Suppose $\left\{10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N+1\right\}$, and it terminates.
- Then (frame rule)

$$\dfrac{\begin{array}{c}\vdots\\ \left\{10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N+1\right\}\end{array}}{\left\{10 \xmapsto[0.5]{} N \star 10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N \star 10 \xmapsto[0.5]{} N+1\right\}}$$

Passivity and fractions

Termination Monotonicity: if C must terminate normally in h and $h \star h'$ is defined, then C must terminate normally in $h \star h'$.

- We can prove termination monotonicity for all commands in our language.
- Suppose $\left\{10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N+1\right\}$, and it terminates.
- Then (frame rule)

$$\dfrac{\begin{array}{c}\vdots\\ \left\{10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N+1\right\}\end{array}}{\left\{10 \xmapsto[0.5]{} N \star 10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N \star 10 \xmapsto[0.5]{} N+1\right\}}$$

- – i.e. it won't terminate in $10 \xmapsto[1.0]{} N$.

Passivity and fractions

Termination Monotonicity: if C must terminate normally in h and $h \star h'$ is defined, then C must terminate normally in $h \star h'$.

- We can prove termination monotonicity for all commands in our language.
- Suppose $\left\{10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N+1\right\}$, and it terminates.
- Then (frame rule)

$$\dfrac{\begin{array}{c}\vdots \\ \left\{10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N+1\right\}\end{array}}{\left\{10 \xmapsto[0.5]{} N \star 10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N \star 10 \xmapsto[0.5]{} N+1\right\}}$$

- – i.e. it won't terminate in $10 \xmapsto[1.0]{} N$.
- Therefore C isn't in our language.

Passivity and fractions

Termination Monotonicity: if C must terminate normally in h and $h \star h'$ is defined, then C must terminate normally in $h \star h'$.

- We can prove termination monotonicity for all commands in our language.
- Suppose $\left\{10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N+1\right\}$, and it terminates.
- Then (frame rule)

$$\frac{\begin{array}{c}\vdots \\ \left\{10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N+1\right\}\end{array}}{\left\{10 \xmapsto[0.5]{} N \star 10 \xmapsto[0.5]{} N\right\} C \left\{10 \xmapsto[0.5]{} N \star 10 \xmapsto[0.5]{} N+1\right\}}$$

- – i.e. it won't terminate in $10 \xmapsto[1.0]{} N$.
- Therefore C isn't in our language.
- Thus we have passivity!

Part II

Counting permissions

Permission counting

Permission counting

- Some programs naturally weigh out permissions to their child threads: e.g. parallel tree-copy, parallel tree-rewriting (see proceedings).

Permission counting

- Some programs naturally weigh out permissions to their child threads: e.g. parallel tree-copy, parallel tree-rewriting (see proceedings).
- Some programs count permissions: e.g. pipeline multicasting, readers-and-writers.

Permission counting

- Some programs naturally weigh out permissions to their child threads: e.g. parallel tree-copy, parallel tree-rewriting (see proceedings).
- Some programs count permissions: e.g. pipeline multicasting, readers-and-writers.
- Permission counting is not specification-only.

Readers and writers (CCR version)

Readers and writers (CCR version)

with *read* when true do
  if $count = 0$ then $\mathrm{P}(write)$
    else skip fi;
  $count{+} := 1$
od;

... reading happens here ...

with *read* when true do
  $count{-} := 1$;
  if $count = 0$ then $\mathrm{V}(write)$
    else skip fi
od

$\mathrm{P}(write)$;

... writing happens here ...

$\mathrm{V}(write)$

Readers and writers (CCR version)

$\{\mathbf{emp}\}$
with read when true do
  if $\mathit{count} = 0$ then $\mathrm{P}(\mathit{write})$
    else skip fi;
  $\mathit{count}{+} := 1$
od;
$\{z \rightarrowtail N\}$
  ... reading happens here ...

with read when true do
  $\mathit{count}{-} := 1;$
  if $\mathit{count} = 0$ then $\mathrm{V}(\mathit{write})$
    else skip fi
od

$\mathrm{P}(\mathit{write});$

  ... writing happens here ...

$\mathrm{V}(\mathit{write})$

Readers and writers (CCR version)

$\{\mathbf{emp}\}$
with *read* when true do
  if $count = 0$ then $\mathrm{P}(write)$
      else skip fi;
  $count{+}:=1$
od;
$\{z \rightarrowtail N\}$
  ... reading happens here ...
$\{z \rightarrowtail N\}$
with *read* when true do
  $count{-}:=1$;
  if $count = 0$ then $\mathrm{V}(write)$
      else skip fi
od
$\{\mathbf{emp}\}$

$\mathrm{P}(write)$;

... writing happens here ...

$\mathrm{V}(write)$

Readers and writers (CCR version)

$\{\mathbf{emp}\}$
with *read* when true do
  if $count = 0$ then $\mathrm{P}(write)$
    else skip fi;
  $count{+} := 1$
od;
$\{z \rightarrowtail N\}$
  ... reading happens here ...
$\{z \rightarrowtail N\}$
with *read* when true do
  $count{-} := 1$;
  if $count = 0$ then $\mathrm{V}(write)$
    else skip fi
od
$\{\mathbf{emp}\}$

$\{\mathbf{emp}\}$
$\mathrm{P}(write)$;
$\left\{z \stackrel{0}{\longmapsto} M\right\}$
  ... writing happens here ...

$\mathrm{V}(write)$

Readers and writers (CCR version)

$\{\mathbf{emp}\}$
with *read* when true do
  if $count = 0$ then $\mathrm{P}(write)$
    else skip fi;
  $count + := 1$
od;
$\{z \rightarrowtail N\}$
  ... reading happens here ...
$\{z \rightarrowtail N\}$
with *read* when true do
  $count - := 1$;
  if $count = 0$ then $\mathrm{V}(write)$
    else skip fi
od
$\{\mathbf{emp}\}$

$\{\mathbf{emp}\}$
$\mathrm{P}(write)$;
$\left\{z \overset{0}{\longmapsto} M\right\}$
  ... writing happens here ...
$\left\{z \overset{0}{\longmapsto} M'\right\}$
$\mathrm{V}(write)$
$\{\mathbf{emp}\}$

A counting model

A counting model

- Heaps are partial maps from Nat to (int, permission).

A counting model

- Heaps are partial maps from Nat to (int, permission).
- Permissions are $-n$ (n read permissions), or $+n$ (a "block" from which n read permissions have been "flaked").

A counting model

- Heaps are partial maps from Nat to (int, permission).
- Permissions are $-n$ (n read permissions), or $+n$ (a “block” from which n read permissions have been “flaked”).
- 0 is total permission.

A counting model

- Heaps are partial maps from Nat to (int, permission).
- Permissions are $-n$ (n read permissions), or $+n$ (a “block” from which n read permissions have been “flaked”).
- 0 is total permission.
- $E \overset{i}{\mapsto} E' \star E \overset{j}{\mapsto} E' = \begin{cases} \text{undefined} & i \geq 0 \wedge j \geq 0 \\ \text{undefined} & (i \geq 0 \vee j \geq 0) \wedge i + j < 0 \\ E \overset{i+j}{\longmapsto} E' & \text{otherwise} \end{cases}$

A counting model

- Heaps are partial maps from Nat to (int, permission).
- Permissions are $-n$ (n read permissions), or $+n$ (a “block” from which n read permissions have been “flaked”).
- 0 is total permission.
- $E \xmapsto{i} E' \star E \xmapsto{j} E' = \begin{cases} \text{undefined} & i \geq 0 \wedge j \geq 0 \\ \text{undefined} & (i \geq 0 \vee j \geq 0) \wedge i + j < 0 \\ E \xmapsto{i+j} E' & \text{otherwise} \end{cases}$
- $E \rightarrowtail E'$ is a notational convenience for $E \xmapsto{-1} E'$.

A counting model

- Heaps are partial maps from Nat to (int, permission).
- Permissions are $-n$ (n read permissions), or $+n$ (a “block” from which n read permissions have been “flaked”).
- 0 is total permission.
- $E \xmapsto{i} E' \star E \xmapsto{j} E' = \begin{cases} \text{undefined} & i \geq 0 \wedge j \geq 0 \\ \text{undefined} & (i \geq 0 \vee j \geq 0) \wedge i + j < 0 \\ E \xmapsto{i+j} E' & \text{otherwise} \end{cases}$
- $E \rightarrowtail E'$ is a notational convenience for $E \xmapsto{-1} E'$.
- We have passivity (same proof as before).

Proof theory

Proof theory

$$E \overset{n}{\longmapsto} E' \quad \Rightarrow \quad n \geq 0$$

$$E \overset{n}{\longmapsto} E' \iff E \xmapsto{n+1} E' \star E \rightarrowtail E'$$

Proof theory

$$E \overset{n}{\longmapsto} E' \quad \Rightarrow \quad n \geq 0$$

$$E \overset{n}{\longmapsto} E' \iff E \overset{n+1}{\longmapsto} E' \star E \rightarrowtail E'$$

$$\begin{array}{lll}
\left\{R_E^x\right\} & x{:=}E & \left\{R\right\} \\
\left\{E' \overset{0}{\longmapsto} _\right\} & [x]{:=}E & \left\{E' \overset{0}{\longmapsto} E\right\} \\
\left\{E' \rightarrowtail E\right\} & x{:=}[E'] & \left\{E' \rightarrowtail E \wedge x = E\right\} \ (x \text{ not free in } E, E') \\
\left\{\mathbf{emp}\right\} & x{:=}\mathrm{new}(E) & \left\{x \overset{0}{\longmapsto} E\right\} \\
\left\{E \overset{0}{\longmapsto} _\right\} & \mathrm{dispose}\, E & \left\{\mathbf{emp}\right\}
\end{array}$$

Resource safety proof

$write$: if $write = 0$ then **emp** else $z \overset{0}{\longmapsto} N$ fi

$read$: if $count = 0$ then **emp** else $z \xmapsto{count} N$ fi

$\{\mathbf{emp}\}$

with $read$ when true do

if $count = 0$ then $\quad$ P($write$)

else $\quad$ skip

fi;

$count+ := 1$

od

$\{z \rightarrowtail N\}$

Resource safety proof

$write : \text{if } write = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{0} N \text{ fi}$

$read : \text{if } count = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{count} N \text{ fi}$

$\{\mathbf{emp}\}$

with $read$ when true do

$\left\{\text{if } count = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{count} N \text{ fi} \star \mathbf{emp}\right\}$

if $count = 0$ then $\qquad \mathrm{P}(write)$

else $\qquad$ skip

fi;

$count{+} := 1$

od

$\{z \rightarrowtail N\}$

Resource safety proof

$write : \text{if } write = 0 \text{ then } \mathbf{emp} \text{ else } z \overset{0}{\longmapsto} N \text{ fi}$

$read : \text{if } count = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{count} N \text{ fi}$

$\{\mathbf{emp}\}$

with $read$ when true do

$\left\{\text{if } count = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{count} N \text{ fi} \star \mathbf{emp}\right\}$

if $count = 0$ then $\{\mathbf{emp}\}$ $\mathrm{P}(write)$

else $\left\{z \xmapsto{count} N\right\}$ skip

fi;

$count{+} := 1$

od

$\{z \rightarrowtail N\}$

Resource safety proof

$write$: if $write = 0$ then $\mathbf{emp}$ else $z \overset{0}{\longmapsto} N$ fi

$read$: if $count = 0$ then $\mathbf{emp}$ else $z \xmapsto{count} N$ fi

$\{\mathbf{emp}\}$

with $read$ when true do

$\left\{\text{if } count = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{count} N \text{ fi} \star \mathbf{emp}\right\}$

if $count = 0$ then $\{\mathbf{emp}\}$ P($write$) $\left\{z \overset{0}{\longmapsto} N\right\}$

else $\left\{z \xmapsto{count} N\right\}$ skip

fi;

$count{+} := 1$

od

$\{z \rightarrowtail N\}$

Resource safety proof

$write : \text{if } write = 0 \text{ then } \mathbf{emp} \text{ else } z \overset{0}{\longmapsto} N \text{ fi}$

$read : \text{if } count = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{count} N \text{ fi}$

$\{\mathbf{emp}\}$

with $read$ when true do

$\left\{\text{if } count = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{count} N \text{ fi} \star \mathbf{emp}\right\}$

if $count = 0$ then $\{\mathbf{emp}\}$ $\mathrm{P}(write)$ $\left\{z \overset{0}{\longmapsto} N\right\}$

else $\left\{z \xmapsto{count} N\right\}$ skip $\left\{z \xmapsto{count} N\right\}$

fi;

$count{+} := 1$

od

$\{z \rightarrowtail N\}$

Resource safety proof

$write: \text{if } write = 0 \text{ then } \mathbf{emp} \text{ else } z \stackrel{0}{\longmapsto} N \text{ fi}$

$read: \text{if } count = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{count} N \text{ fi}$

$\{\mathbf{emp}\}$

with *read* when true do

$\left\{\text{if } count = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{count} N \text{ fi} \star \mathbf{emp}\right\}$

if $count = 0$ then $\{\mathbf{emp}\}$ P$(write)$ $\left\{z \stackrel{0}{\longmapsto} N\right\}$

else $\left\{z \xmapsto{count} N\right\}$ skip $\left\{z \xmapsto{count} N\right\}$

fi;

$\left\{z \xmapsto{count} N\right\}$

$count{+} := 1$

od

$\{z \rightarrowtail N\}$

Resource safety proof

$write$: if $write = 0$ then $\mathbf{emp}$ else $z \overset{0}{\longmapsto} N$ fi

$read$: if $count = 0$ then $\mathbf{emp}$ else $z \xmapsto{count} N$ fi

$\{\mathbf{emp}\}$

with $read$ when true do

$\left\{\text{if } count = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{count} N \text{ fi} \star \mathbf{emp}\right\}$

if $count = 0$ then $\{\mathbf{emp}\}$ P($write$) $\left\{z \overset{0}{\longmapsto} N\right\}$

else $\left\{z \xmapsto{count} N\right\}$ skip $\left\{z \xmapsto{count} N\right\}$

fi;

$\left\{z \xmapsto{count} N\right\}$

$count+ := 1$

$\left\{z \xmapsto{count-1} N\right\}$

od

$\{z \rightarrowtail N\}$

Resource safety proof

$$write : \text{if } write = 0 \text{ then } \mathbf{emp} \text{ else } z \overset{0}{\longmapsto} N \text{ fi}$$
$$read : \text{if } count = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{count} N \text{ fi}$$

$\{\mathbf{emp}\}$
with *read* when true do

$\left\{\text{if } count = 0 \text{ then } \mathbf{emp} \text{ else } z \xmapsto{count} N \text{ fi} \star \mathbf{emp}\right\}$

if $count = 0$ then $\quad \{\mathbf{emp}\}\ \mathrm{P}(write)\ \left\{z \overset{0}{\longmapsto} N\right\}$

else $\left\{z \xmapsto{count} N\right\}$ skip $\left\{z \xmapsto{count} N\right\}$

fi;

$\left\{z \xmapsto{count} N\right\}$

$count+ := 1$

$\left\{z \xmapsto{count-1} N\right\} \therefore \left\{z \xmapsto{count} N \star z \rightarrowtail N\right\}$

od

$\{z \rightarrowtail N\}$

Something you may have missed ...

Something you may have missed ...

- Readers-and-writers has several distinct parts: reader prologue, reader action, reader epilogue, writer P, writer action, writer V.

Something you may have missed ...

- Readers-and-writers has several distinct parts: reader prologue, reader action, reader epilogue, writer P, writer action, writer V.
- $$\begin{array}{l} prologue; prologue; prologue; \\ \left(\begin{array}{l} reader_1; \\ epilogue \end{array} \middle\| \, reader_2 \, \middle\| \, reader_3 \right); \\ epilogue; reader_4; epilogue \end{array}$$

Something you may have missed ...

- Readers-and-writers has several distinct parts: reader prologue, reader action, reader epilogue, writer P, writer action, writer V.
- $$\begin{array}{l} prologue; prologue; prologue; \\ \left(\begin{array}{l} reader_1; \\ epilogue \end{array} \middle\| reader_2 \middle\| reader_3 \right); \\ epilogue; reader_4; epilogue \end{array}$$
- $$\mathrm{P}(write); writer_1; \left(reader_5 \,\middle\|\, reader_6 \right); writer_2; \mathrm{V}(write)$$

Something you may have missed ...

- Readers-and-writers has several distinct parts: reader prologue, reader action, reader epilogue, writer P, writer action, writer V.
- $$\begin{array}{l} \mathit{prologue}; \mathit{prologue}; \mathit{prologue}; \\ \left(\begin{array}{l} \mathit{reader}_1; \\ \mathit{epilogue} \end{array} \middle\| \; \mathit{reader}_2 \; \middle\| \; \mathit{reader}_3 \right); \\ \mathit{epilogue}; \mathit{reader}_4; \mathit{epilogue} \end{array}$$
- $$\mathrm{P}(\mathit{write}); \mathit{writer}_1; \left(\mathit{reader}_5 \middle\| \mathit{reader}_6 \right); \mathit{writer}_2; \mathrm{V}(\mathit{write})$$
- No more critical sections! – now *that's* resourcing!

We do have some unsolved problems ...

We do have some unsolved problems ...

$$
\begin{aligned}
\mathrm{ztree}\ z\ \mathrm{nil}\ \mathsf{Empty} &\mathrel{\hat{=}} \mathbf{emp} \\
\mathrm{ztree}\ z\ t\ (\mathsf{Tip}\ \alpha) &\mathrel{\hat{=}} t \underset{z}{\longmapsto} 0, \alpha \\
\mathrm{ztree}\ z\ t\ (\mathsf{Node}\ \lambda\ \rho) &\mathrel{\hat{=}} \exists l, r \cdot (t \underset{z}{\longmapsto} 1, l, r \star \mathrm{ztree}\ z\ l\ \lambda \star \mathrm{ztree}\ z\ r\ \rho)
\end{aligned}
$$

We do have some unsolved problems ...

$$\begin{aligned}
\mathrm{ztree}\ z\ \mathrm{nil}\ \mathsf{Empty} &\mathrel{\hat{=}} \mathbf{emp}\\
\mathrm{ztree}\ z\ t\ (\mathsf{Tip}\ \alpha) &\mathrel{\hat{=}} t \underset{z}{\mapsto} 0, \alpha\\
\mathrm{ztree}\ z\ t\ (\mathsf{Node}\ \lambda\ \rho) &\mathrel{\hat{=}} \exists l, r \cdot (t \underset{z}{\mapsto} 1, l, r \star \mathrm{ztree}\ z\ l\ \lambda \star \mathrm{ztree}\ z\ r\ \rho)
\end{aligned}$$

- $\mathrm{ztree}\ 0.5\ t\ (\mathsf{Node}\ (\mathsf{Tip}\ 4)\ (\mathsf{Tip}\ 4))$ isn’t necessarily separated: it could be a DAG.

We do have some unsolved problems ...

$$
\begin{aligned}
\mathrm{ztree}\ z\ \mathrm{nil}\ \mathsf{Empty} &\mathrel{\hat{=}} \mathbf{emp}\\
\mathrm{ztree}\ z\ t\ (\mathsf{Tip}\ \alpha) &\mathrel{\hat{=}} t \underset{z}{\hookrightarrow} 0, \alpha\\
\mathrm{ztree}\ z\ t\ (\mathsf{Node}\ \lambda\ \rho) &\mathrel{\hat{=}} \exists l, r \cdot (t \underset{z}{\hookrightarrow} 1, l, r \star \mathrm{ztree}\ z\ l\ \lambda \star \mathrm{ztree}\ z\ r\ \rho)
\end{aligned}
$$

- $\mathrm{ztree}\ 0.5\ t\ (\mathsf{Node}\ (\mathsf{Tip}\ 4)\ (\mathsf{Tip}\ 4))$ isn't necessarily separated: it could be a DAG.
- We have two models – fractions and counting. Both seem to be necessary at present (maybe ...).

Future work

Future work

- Variables as resources.

Future work

- Variables as resources.
- Existence permissions.

Future work

- Variables as resources.
- Existence permissions.
- Semaphores in the heap.

Future work

- Variables as resources.
- Existence permissions.
- Semaphores in the heap.
- Soundness (we just need a nod ...)

Accounting for variables

Accounting for variables

- $$\frac{\{Q\}C\{R\}}{\{P \star Q\}C\{P \star R\}}\ (\mathit{modifies}\ C \cap \mathit{vars}\ P = \emptyset).$$

Accounting for variables

- $$\frac{\{Q\}C\{R\}}{\{P \star Q\}C\{P \star R\}} \quad (\textit{modifies } C \cap \textit{vars } P = \emptyset).$$

  $\{\mathbf{emp}\}$
  $\quad \mathrm{P}(\textit{read});$
  $\{\mathrm{own}(\textit{count}) \wedge \ldots\}$
  if $\textit{count} = 0$ then ... // safe: I can read *count*

- $\left\{\mathrm{own}(\textit{count}) \wedge z \xmapsto{\textit{count}} N\right\}$
  $\quad \textit{count} + := 1;$ // safe: I can read and write *count*
  $\{\mathrm{own}(\textit{count}) \wedge z \xmapsto{\textit{count}} N \star z \rightarrowtail N\}$
  $\quad \mathrm{V}(\textit{read})$
  $\{z \rightarrowtail N\}$
  ... // no longer safe to read or write *count*

Accounting for variables

- $$\frac{\{Q\}C\{R\}}{\{P \star Q\}C\{P \star R\}}\ (\textit{modifies}\ C \cap \textit{vars}\ P = \emptyset).$$

- $\{\mathbf{emp}\}$
  $\mathrm{P}(\textit{read});$
  $\{\mathrm{own}(\textit{count}) \wedge ...\}$
  if $\textit{count} = 0$ then ... // safe: I can read $\textit{count}$
  $\left\{\mathrm{own}(\textit{count}) \wedge z \xmapsto{\textit{count}} N\right\}$
  $\textit{count} + := 1;$ // safe: I can read and write $\textit{count}$
  $\{\mathrm{own}(\textit{count}) \wedge z \xmapsto{\textit{count}} N \star z \rightarrowtail N\}$
  $\mathrm{V}(\textit{read})$
  $\{z \rightarrowtail N\}$
  ... // no longer safe to read or write $\textit{count}$

- Imagine: no non-interference side-conditions; no anti-aliasing side-condition on variable assignment; elegant proofs of Dijkstra's semaphore programs – now *that's* hubris!

Is anybody there?

Is anybody there?

- A semaphore has permission to read and write its variable; a user has permission to P and V it.

Is anybody there?

- A semaphore has permission to read and write its variable; a user has permission to P and V it.
- So long as any user has that permission, the semaphore can't dispose itself.

Is anybody there?

- A semaphore has permission to read and write its variable; a user has permission to P and V it.
- So long as any user has that permission, the semaphore can't dispose itself.
- Existence permissions give no access to resource contents.

Is anybody there?

- A semaphore has permission to read and write its variable; a user has permission to P and V it.
- So long as any user has that permission, the semaphore can't dispose itself.
- Existence permissions give no access to resource contents.
- Proof theory looks easy, but no agreed model yet.

Last one to leave, turn off the light!

Last one to leave, turn off the light!

- Pipeline processing fits permission counting: e.g. multicasting in a network processor.

Last one to leave, turn off the light!

- Pipeline processing fits permission counting: e.g. multicasting in a network processor.
- Each packet-buffer has a semaphore, in neighbouring heap location, which counts permissions.

Last one to leave, turn off the light!

- Pipeline processing fits permission counting: e.g. multicasting in a network processor.
- Each packet-buffer has a semaphore, in neighbouring heap location, which counts permissions.
- ‘start’ gives you a buffer and a 1 semaphore.

Last one to leave, turn off the light!

- Pipeline processing fits permission counting: e.g. multicasting in a network processor.
- Each packet-buffer has a semaphore, in neighbouring heap location, which counts permissions.
- ‘start’ gives you a buffer and a 1 semaphore.
- ‘split’ Vs the semaphore, and releases an extra read permission.

Last one to leave, turn off the light!

- Pipeline processing fits permission counting: e.g. multicasting in a network processor.
- Each packet-buffer has a semaphore, in neighbouring heap location, which counts permissions.
- ‘start’ gives you a buffer and a 1 semaphore.
- ‘split’ Vs the semaphore, and releases an extra read permission.
- ‘finish’ at the end of a pipeline Ps the semaphore – and if now 0, disposes semaphore and buffer together.

Last one to leave, turn off the light!

- Pipeline processing fits permission counting: e.g. multicasting in a network processor.
- Each packet-buffer has a semaphore, in neighbouring heap location, which counts permissions.
- ‘start’ gives you a buffer and a 1 semaphore.
- ‘split’ Vs the semaphore, and releases an extra read permission.
- ‘finish’ at the end of a pipeline Ps the semaphore – and if now 0, disposes semaphore and buffer together.
- The semaphore has to be hidden; it might need a CCR; it might need the hypothetical frame rule; it’s a semaphore in the heap!...

Soundness

We think we only need a nod.

Thank you!

John Tang Boyland, for pointing out that $\frac{1}{2} + \frac{1}{2} = 1$.

The East London Massive for listening to countless versions, and picking holes in them.

Steve Brookes, for taking us seriously.

Josh Berdine, John Reynolds and Hongseok Yang, for ruthless and relentless criticism.