Concurrent separation logic

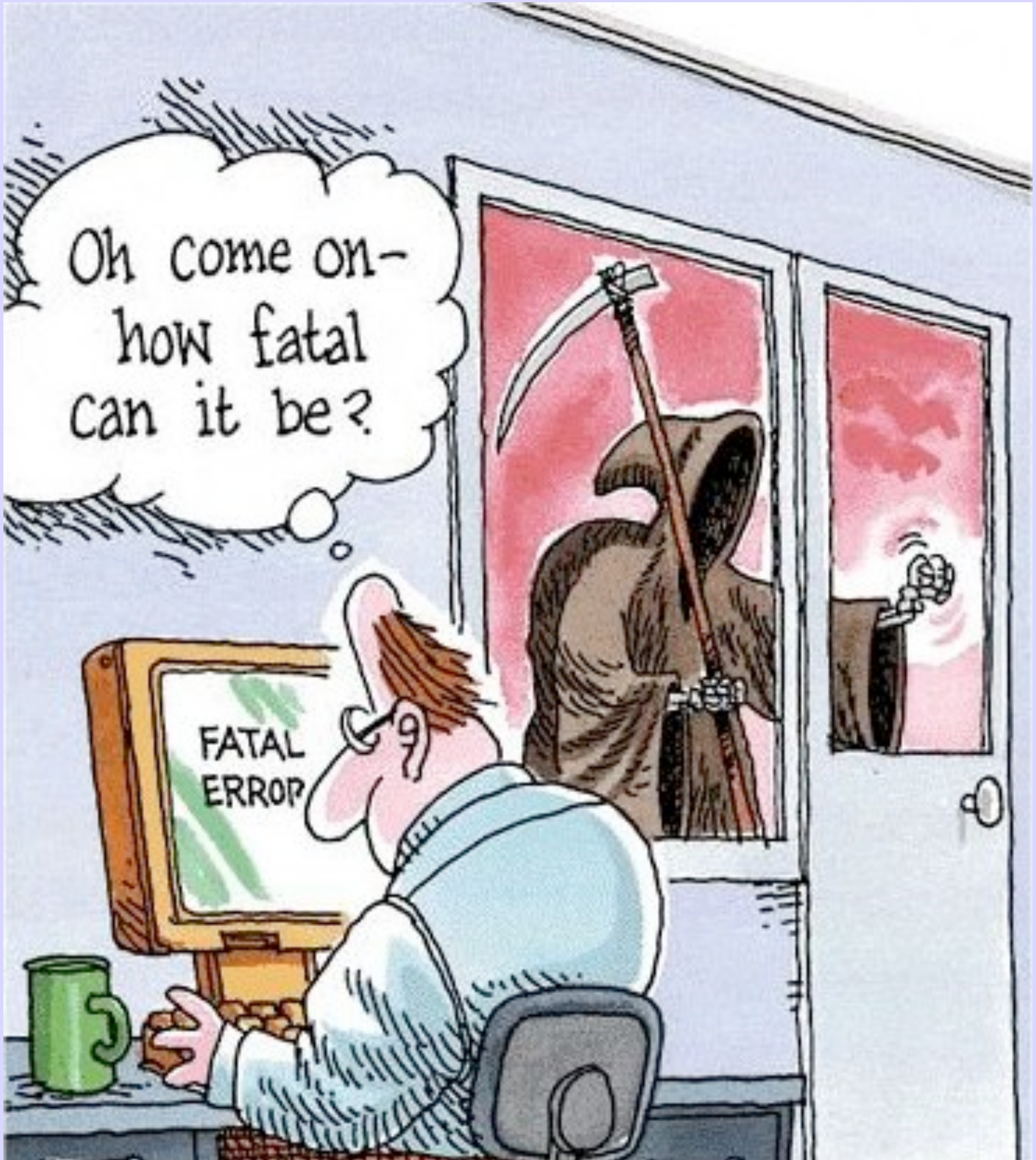
Stephen Brookes
Carnegie Mellon University


Programs

- Imperative, using pointers
  - *mutable data*
- Concurrent execution
  - *shared state*
- Synchronization
  - *mutual exclusion*

Problems

- *Concurrent* programs are hard to get right
  - *race conditions, deadlock, mutual exclusion*
- Even *sequential* pointer programs can be tricky
  - *dangling pointers*
- Traditional methods don't work...
  - *pointers + concurrency* $\Rightarrow$ *no static checking*


Oh come on-
how fatal
can it be?
FATAL
ERROR

Race conditions

... cause unpredictable behavior
... results may depend on granularity

- Concurrent write

  x:=1 || x:=2

- Concurrent update

  [x]:=1 || [y]:=2

- Concurrent disposal

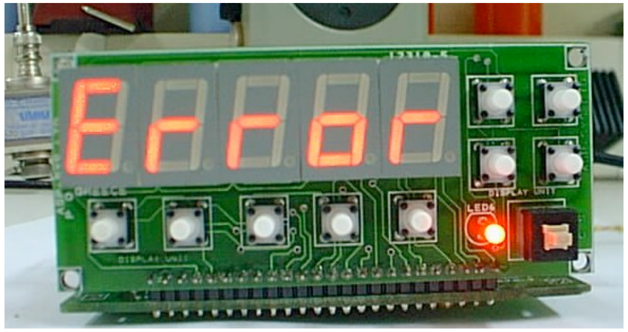
  dispose x || **dispose** y

Outline

- A *denotational* semantic model
  - *syntax-directed*
  - *race-detecting*
- Support for *compositional* reasoning
  - *concurrent separation logic*
  - *race-avoiding program correctness*
- Advantages of approach
  - *local reasoning improves scalability*

Semantic model

- A command denotes a set of *action traces*
  - *trace = sequence of actions*
- *Actions* have *effect* on *state*
  - *state = store + heap + resources*
- Traces describe *interactive computations*
  - *fair, resource-sensitive, race-detecting*



Actions

- δ — idle
- $i{=}v,\ \ i{:=}v$ — read, write
- $[v]{=}v',\ \ [v]{:=}v'$ — lookup, update
- $alloc(v, L),\ \ disp(v)$ — allocate, dispose
- $try(r),\ \ acq(r),\ \ rel(r)$ — try, acquire, release
- $abort$ — runtime error

λ ranges over actions

Traces

- trace = sequence of actions
    - *finite or infinite*
- concatenation
    - $\alpha\,\delta\,\beta = \alpha\,\beta$
    - α *abort* $\beta = \alpha$ *abort*

α, β *range over traces*

Tr *is the set of all traces*

Semantic functions

- Integer expressions

$$[\![e]\!] \subseteq \mathbf{Tr} \times V$$

- Boolean expressions

$$[\![b]\!]_{true}, [\![b]\!]_{false} \subseteq \mathbf{Tr}$$

- List expressions

$$[\![E]\!] \subseteq \mathbf{Tr} \times V^*$$

- Commands

$$[\![c]\!] \subseteq \mathbf{Tr}$$

... defined denotationally

Semantic clauses

$$[\![\mathbf{skip}]\!] = \{\delta\}$$

$$[\![i{:=}e]\!] = \{\rho\, i{:=}v \mid (\rho, v) \in [\![e]\!]\}$$

$$[\![c_1; c_2]\!] = [\![c_1]\!]\,[\![c_2]\!]$$

$$[\![\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2]\!] = [\![b]\!]_{true}\,[\![c_1]\!] \cup [\![b]\!]_{false}\,[\![c_2]\!]$$

$$[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!] = ([\![b]\!]_{true}\,[\![c]\!])^{*}\,[\![b]\!]_{false} \cup ([\![b]\!]_{true}\,[\![c]\!])^{\omega}$$

sequential constructs

Semantic clauses

$$[\![i{:=}[e]]\!] = \{\rho\,[v]{=}v'\,i{:=}v' \mid (\rho, v) \in [\![e]\!]\}$$

$$[\![i{:=}\mathbf{cons}(E)]\!] = \{\rho\,alloc(l, L)\,i{:=}l \mid (\rho, L) \in [\![E]\!]\}$$

$$[\![[e]{:=}e']\!] = \{\rho\,\rho'\,[v]{:=}v' \mid (\rho, v) \in [\![e]\!] \;\&\; (\rho', v') \in [\![e']\!]\}$$

$$[\![\mathbf{dispose}\,e]\!] = \{\rho\,disp\,l \mid (\rho, l) \in [\![e]\!]\}$$

pointer operations

Synchronization

$$[\![\mathbf{with}\ r\ \mathbf{when}\ b\ \mathbf{do}\ c]\!] = wait^*\ enter\ \cup\ wait^\omega$$

where

$$wait = \{try\, r\}\ \cup\ acq\, r\, [\![b]\!]_{false}\, rel\, r$$

$$enter = acq\, r\, [\![b]\!]_{true}\, [\![c]\!]\, rel\, r$$

conditional critical region

Local variable

$$[\![\textbf{local } i = e \textbf{ in } c]\!] = \{\rho\,(\alpha \backslash i) \mid (\rho, v) \in [\![e]\!] \ \& \ \alpha \in [\![c]\!]_{i=v}\}$$

$\alpha \restriction i$ *executable* from [*i*:*v*]

local variable *i*
only accessible inside *c*

statically scoped block

Local resource

$$[\![\mathbf{resource}\ r\ \mathbf{in}\ c]\!] = \{\alpha \backslash r \mid \alpha \in [\![c]\!]_r\}$$

$\alpha \upharpoonright r$ *executable* from { }

local resource *r*
only accessible inside *c*

statically scoped

Parallel composition

$$[\![c_1 \| c_2]\!] = [\![c_1]\!]_{\{\}} \|_{\{\}} [\![c_2]\!]$$

- processes start with no resources
- resources are mutually exclusive
- race produces error

Ingredients

- What a process can do depends on its resources and those of its environment

$$(A_1, A_2) \xrightarrow{\lambda} (A_1', A_2)$$

resource enabling relation

- A race is interpreted as an error

$$\lambda_1 \bowtie \lambda_2$$

interfering actions

Resource enabling

$$(A_1, A_2) \xrightarrow{\lambda} (A_1', A_2)$$

process with A_1 *can do* λ *in environment with* A_2

$$(A_1, A_2) \xrightarrow{acq\, r} (A_1 \cup \{r\}, A_2) \quad \text{if } r \notin A_1 \cup A_2$$

$$(A_1, A_2) \xrightarrow{rel\, r} (A_1 - \{r\}, A_2) \quad \text{if } r \in A_1$$

$$(A_1, A_2) \xrightarrow{\lambda} (A_1, A_2) \quad \text{if } r \neq acq\, r,\ rel\, r$$

Interfering actions

... one writes to variable or heap cell used by the other

$$\lambda_1 \bowtie \lambda_2$$

iff

$$free(\lambda_1) \cap writes(\lambda_2) \neq \{\}$$

or

$$free(\lambda_2) \cap writes(\lambda_1) \neq \{\}$$

Interleaving

... fair, resource-sensitive, race-detecting

$$\alpha \,{}_{A_1}\|_{A_2}\, \epsilon \;=\; \{\alpha \mid (A_1, A_2) \xrightarrow{\alpha} \cdot\}$$

$$\epsilon \,{}_{A_1}\|_{A_2}\, \alpha \;=\; \{\alpha \mid (A_2, A_1) \xrightarrow{\alpha} \cdot\}$$

$$(\lambda_1\alpha_1)\,{}_{A_1}\|_{A_2}\,(\lambda_2\alpha_2) \;=$$

$$\{\lambda_1\beta \mid (A_1, A_2) \xrightarrow{\lambda_1} (A_1', A_2) \;\&\; \beta \in \alpha_1\,{}_{A_1'}\|_{A_2}\,(\lambda_2\alpha_2)\}$$

$$\cup\; \{\lambda_2\beta \mid (A_2, A_1) \xrightarrow{\lambda_2} (A_2', A_1) \;\&\; \beta \in (\lambda_1\alpha_1)\,{}_{A_1}\|_{A_2'}\,\alpha_2\}$$

$$\cup\; \{abort \mid \lambda_1 \bowtie \lambda_2\}$$

Examples

$$[\![x{:=}1 \| y{:=}1]\!] \;=\; \{x{:=}1\, y{:=}1,\; y{:=}1\, x{:=}1\}$$

$$[\![x{:=}1 \| x{:=}1]\!] \;=\; \{x{:=}1\, x{:=}1,\; abort\}$$

$$[\![\textbf{with}\ r\ \textbf{do}\ x{:=}1]\!] \;=\; (try\, r)^*\, acq\, r\, x{:=}1\, rel\, r \;\cup\; (try\ r)^\omega$$

$$[\![\textbf{resource}\ r\ \textbf{in}\ (\textbf{with}\ r\ \textbf{do}\ x{:=}1 \| \textbf{with}\ r\ \textbf{do}\ x{:=}1)]\!]$$

$$=\; \{x{:=}1\, x{:=}1\}$$

PUT

$$\textbf{with } \mathit{buf} \textbf{ when } \neg \mathit{full} \textbf{ do}$$
$$(c{:=}x;\ \mathit{full}{:=}\textbf{true})$$

Typical trace

$$(\mathit{acq}\ \ \mathit{buf}\,)\,\mathit{full}{=}\mathit{false}\,\mathrm{put}\,v\,(\mathit{rel}\ \mathit{buf}\,)$$

where

$$\mathrm{put}\,v \ \ =_{def}\ \ x{=}v\,c{:=}v\,\mathit{full}{:=}\mathit{true}$$

GET

$$\textbf{with } buf \textbf{ when } full \textbf{ do}$$
$$(y{:=}c;\ full{:=}\textbf{false})$$

Typical trace

$$(acq\ buf\,)\,full{=}true\,\mathrm{get}\,v'\,(rel\ buf\,)$$

where

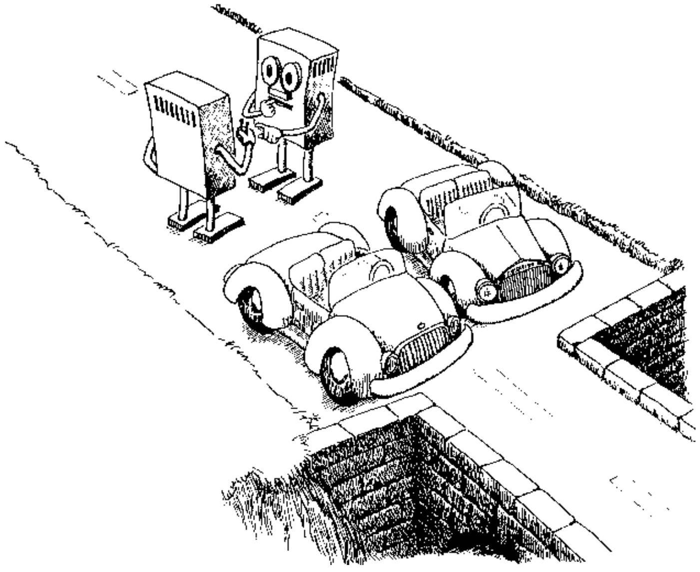
$$\mathrm{get}\,v' \;=_{def}\; c{=}v'\,y{:=}v'\,full{:=}false$$

Deadlock

$$\begin{array}{l}\textbf{resource } r_1, r_2 \textbf{ in} \\ \quad\quad \textbf{with } r_1 \textbf{ do with } r_2 \textbf{ do } x{:=}1 \\ \quad \| \ \textbf{with } r_2 \textbf{ do with } r_1 \textbf{ do } y{:=}1\end{array}$$

has trace set

$$\{x{:=}1\, y{:=}1,\ y{:=}1\, x{:=}1,\ \delta^\omega\}$$



Process state

$$(s, h, A)$$

- global store $s \ : \mathbf{Ide} \rightharpoonup V$
- global heap $h \ : \mathbf{Loc} \rightharpoonup V$
- resources A held by process

Effects

- State may *enable* action by process
- Action causes *state change*

$$(s, h, A) \xRightarrow{\lambda} (s', h', A')$$

$$(s, h, A) \xRightarrow{\lambda} \mathbf{abort}$$

defined by cases

Store actions

$$(s, h, A) \overset{\delta}{\Longrightarrow} (s, h, A)$$

idle

$$(s, h, A) \overset{i=v}{\Longrightarrow} (s, h, A) \quad \mathit{if}\ (i, v) \in s$$

read

$$(s, h, A) \overset{i:=v}{\Longrightarrow} ([s \mid i : v], h, A)$$

$$\mathit{if}\ i \in \mathit{dom}\, s$$

write

Heap actions

$$(s, h, A) \xrightarrow{[v]=v'} (s, h, A) \quad if\ (v, v') \in h$$ lookup

$$(s, h, A) \xrightarrow{[v]:=v'} (s, [h \mid v : v'], A)$$ update

$$if\ v \in dom\, h$$

allocate

$$(s, h, A) \xrightarrow{alloc(v,[v_0,\ldots,v_n])} (s, [h \mid v : v_0, \ldots, v+n : v_n], A)$$

$$if\ v, v+1, \ldots, v+n \notin dom\, h$$

$$(s, h, A) \xrightarrow{disp\ v} (s, h \backslash v, A)$$ dispose

$$if\ v \in dom\, h$$

Resource actions

$$(s, h, A) \xRightarrow{acq\ r} (s, h, A \cup \{r\}) \quad if\ r \notin A$$ acquire

$$(s, h, A) \xRightarrow{rel\ r} (s, h, A - \{r\}) \quad if\ r \in A$$ release

$$(s, h, A) \xRightarrow{try\ r} (s, h, A)$$ try

Errors

$$(s, h, A) \xRightarrow{i=v} \mathbf{abort}$$

$$(s, h, A) \xRightarrow{i:=v} \mathbf{abort} \qquad \textit{if } \; i \notin \textit{dom}\, s$$

$$(s, h, A) \xRightarrow{[v]=v'} \mathbf{abort}$$

$$(s, h, A) \xRightarrow{[v]:=v'} \mathbf{abort}$$

$$(s, h, A) \xRightarrow{\textit{disp}\ v} \mathbf{abort} \qquad \textit{if } \; v \notin \textit{dom}\, h$$

$$(s, h, A) \xRightarrow{\textit{abort}} \mathbf{abort}$$

$$\mathbf{abort} \xRightarrow{\lambda} \mathbf{abort}$$

Computation

- An *executable* sequence of actions
  - no interference between steps

$$(s, h, A) \stackrel{\alpha}{\Longrightarrow} (s', h', A')$$

$$(s, h, A) \stackrel{\alpha}{\Longrightarrow} \mathbf{abort}$$

defined by composition

A computation

of PUT || (GET; dispose y)

$$([x:v, y:\_, full:false, c:\_],\ [v:\_],\ \{\})$$

$$\xrightarrow{acq\ buf}\quad ([x:v, y:\_, full:false, c:\_],\ [v:\_],\ \{buf\})$$

$$\xrightarrow{full=false\ \mathrm{put}\ v}\quad ([x:v, y:\_, full:true, c:v],\ [v:\_],\ \{buf\})$$

$$\xrightarrow{rel\ buf}\quad ([x:v, y:\_, full:true, c:v],\ [v:\_],\ \{\})$$

$$\xrightarrow{acq\ buf}\quad ([x:v, y:\_, full:true, c:v],\ [v:\_],\ \{buf\})$$

$$\xrightarrow{full=true\ \mathrm{get}\ v}\quad ([x:v, y:v, full:false, c:v],\ [v:\_],\ \{buf\})$$

$$\xrightarrow{rel\ buf}\quad ([x:v, y:v, full:false, c:v],\ [v:\_],\ \{\})$$

$$\xrightarrow{y=v\ disp\ v}\quad ([x:v, y:v, full:false, c:v],\ [\,],\ \{\})$$

Error-free

Definition

c is error-free from (s,h)

iff

$$\forall \alpha \in [\![c]\!].\ \neg((s,h) \overset{\alpha}{\Longrightarrow} \mathbf{abort})$$

EXAMPLE

$$\mathbf{dispose}\, x \,\|\, \mathbf{dispose}\, y$$

is error-free iff

$$s(x) \neq s(y)\ \&\ s(x), s(y) \in dom\, h$$


STOP THIS
WE WANT RACE-FREE PROGRAMS
NO DANGLING POINTERS

Examples

- PUT || (GET; dispose y)
  error-free if $s(full) = true$ & $s(c) \in dom(h)$
  or $s(full) = false$ & $s(x) \in dom(h)$
- (PUT; dispose x) || GET
  error-free if $s(full) \in \{true, false\}$ & $s(x) \in dom(h)$
- (PUT; dispose x) || (GET; dispose y)
  is not error-free

So far...

- Trace semantics
  - *denotational, hence compositional*
  - *race-detecting*
- Designed to support program analysis
  - *partial, total correctness*
  - *safety and liveness*

Next: a logic

- Based on trace semantics
- Syntax-directed inference rules
- Resource-sensitive partial correctness
  - *with guaranteed race-freedom*

Traditionally

... Hoare, Owicki-Gries, Dijkstra

- Partition the *critical variables*
  - among resources and processes
- Encapsulate information in resource invariants
  - expressed with first-order logic
- Inference rules enforce discipline
  - *conjunction* of resource invariants
  - *mutual exclusion for critical variables*

Traditionally

... Hoare, Owicki-Gries, Dijkstra

NOT SOUND FOR POINTER PROGRAMS

$$\frac{\Gamma \vdash \{p_1\}c_1\{q_1\} \quad \Gamma \vdash \{p_2\}c_2\{q_2\}}{\Gamma \vdash \{p_1 \wedge p_2\}c_1 \| c_2\{q_1 \wedge q_2\}}$$

provided ...

$$\frac{\Gamma \vdash \{(p \wedge R) \wedge b\}c\{q \wedge R\}}{\Gamma, r(X){:}R \vdash \{p\}\mathbf{with}\ r\ \mathbf{when}\ b\ \mathbf{do}\ c\{q\}}$$

$$\frac{\Gamma, r(X){:}R \vdash \{p\}c\{q\}}{\Gamma \vdash \{p \wedge R\}\mathbf{resource}\ r\ \mathbf{in}\ c\{q \wedge R\}}$$

Generalization

... *O'Hearn*

- Partition the *critical variables and heap*
  - among resources and processes
- Encapsulate information in resource invariants
  - expressed with *separation logic*
- Inference rules enforce discipline
  - *separate conjunction* of resource invariants
  - *mutual exclusion for critical variables and heap*

Separation logic

... Reynolds

separating conjunction

$\vartheta ::= p \mid emp \mid e \mapsto e' \mid \vartheta_1 * \vartheta_2 \mid \vartheta_1 \wedge \vartheta_2 \ldots$

- e, e' range over *pure* integer expressions
- p ranges over *pure* boolean expressions
- formulas describe store + heap

$$(s,h) \models \vartheta$$

(pure = independent of heap)

Satisfaction

- $(s,h) \models p$ iff $|p|s = \mathit{true}$
- $(s,h) \models \mathsf{emp}$ iff $h = \{\ \}$
- $(s,h) \models e \mapsto e'$ iff $h = \{(|e|s,\ |e'|s)\}$
- $(s,h) \models \vartheta_1 * \vartheta_2$ iff

$$\exists h_1 \perp h_2.\ h = h_1 \cdot h_2 \ \&\ (s,h_1) \models \vartheta_1 \ \&\ (s,h_2) \models \vartheta_2$$

ϑ_1 and ϑ_2 hold separately

Resource contexts

$$\Gamma ::= r_1(X_1){:}R_1, \ldots, r_n(X_n){:}R_n$$

satisfying *modularity properties*

$$i \neq j \;\Rightarrow\; X_i \cap X_j = \{\}$$

$$i \neq j \;\Rightarrow\; \mathit{free}(R_i) \cap X_j = \{\}$$

- resource names $\quad dom\,\Gamma = \{r_1, \ldots, r_n\}$
- protection lists $\quad owned\,\Gamma = X_1 \cup \cdots \cup X_n$
- invariants $\quad inv\,\Gamma = R_1 \star \cdots \star R_n$

Precision

Each invariant must be precise

R is *precise* if
  for all states (s, h)
    there is at most one $h' \subseteq h$
    such that
      $(s, h') \models R$

A resource invariant uniquely determines a sub-heap

Precision

- emp is precise
- $e \mapsto e'$ is precise
- if ϑ_1 and ϑ_2 are precise, so is $\vartheta_1 * \vartheta_2$
- if ϑ is precise, and p is pure, $p \wedge \vartheta$ is precise
- if ϑ_1 and ϑ_2 are precise, and b is pure,

  $(b \wedge \vartheta_1) \vee (\neg b \wedge \vartheta_2)$ is precise

Specifications

$$\Gamma \vdash \{p\}c\{q\}$$

Well-formed when

- critical variables of c are protected in Γ
- c reads/writes protected variables inside region
- c writes free variables of invariants inside region
- p and q don't mention protected variables

$$free(p, q) \cap owned\,\Gamma = \{\}$$

Properties enforced by the inference rules

Validity of $\Gamma \vdash \{p\}c\{q\}$?

The obvious candidate definition...

Every finite *computation* of c
from a state satisfying $p \star inv\,\Gamma$

is error-free,
and ends in a state satisfying $q \star inv\,\Gamma$

NOT COMPOSITIONAL
(ignores interaction)

Validity of $\Gamma \vdash \{p\}c\{q\}$

An informal working definition...

Every finite *interactive computation* of c
in an environment that respects Γ
from a state satisfying $p \star inv\,\Gamma$

is error-free, respects Γ,
and ends in a state satisfying $q \star inv\,\Gamma$

... *COMPOSITIONAL*
(a form of rely/guarantee)

Inference rules

based on

- Hoare, Owicki-Gries
  - *concurrency, no pointers*
- Reynolds, O'Hearn
  - *pointers, no concurrency*
- O'Hearn

$\wedge \quad \mapsto \quad \star$

a simple trick
with deep ramifications

assignment

$$\frac{}{\Gamma \vdash \{[e/i]p\} i := e \{p\}}$$

$$\text{if } \; i \notin \mathit{owned}\,\Gamma \cup \mathit{free}\,\Gamma$$

$$\text{and } \; \mathit{free}(p, e) \cap \mathit{owned}\,\Gamma = \{\}$$

cf. Hoare logic,
sequential separation logic

lookup

$$\frac{}{\Gamma \vdash \{[e'/i]p \wedge e \mapsto e'\}\, i := [e]\, \{p \wedge e \mapsto e'\}}$$

$$if\ \ i \notin owned\,\Gamma \cup free\,\Gamma$$

$$and\ \ i \notin free(e, e')$$

$$and\ \ free(p, e, e') \cap owned\,\Gamma = \{\}$$

cf. sequential separation logic

update

$$\frac{}{\Gamma \vdash \{e \mapsto \_\}[e]{:=}e'\{e \mapsto e'\}}$$

$$\mathit{if}\ \ \mathit{free}(e, e') \cap \mathit{owned}\,\Gamma = \{\}$$

$e \mapsto \_ \ =_{def} \exists x.\ e \mapsto x$

cf. sequential separation logic

allocation

$$\overline{\Gamma \vdash \{\mathbf{emp}\} i := \mathbf{cons}(E) \{ i \mapsto E \}}$$

$$if \;\; i \notin free(E)$$

$$and \;\; free(E) \cap owned\,\Gamma = \{\}$$

$$and \;\; i \notin owned\,\Gamma \cup free\,\Gamma$$

$e \mapsto [e_0, \ldots, e_n]$

$=_{def} e \mapsto e_0 * e+1 \mapsto e_1 * e+n \mapsto e_n$

disposal

$$\overline{\Gamma \vdash \{e \mapsto \_\}\, \mathbf{dispose}\, e \{\mathbf{emp}\}}$$

$$\textit{if}\ \ \textit{free}(e) \cap \textit{owned}\, \Gamma = \{\}$$

IMPORTANT:
axioms for heap ops
are “tight”

parallel

$*$ instead of $\wedge$

$$\frac{\Gamma \vdash \{p_1\} c_1 \{q_1\} \qquad \Gamma \vdash \{p_2\} c_2 \{q_2\}}{\Gamma \vdash \{p_1 \star p_2\} c_1 \| c_2 \{q_1 \star q_2\}}$$

if

$$free(c_1) \cap writes(c_2) \subseteq owned\,\Gamma$$

$$free(c_2) \cap writes(c_1) \subseteq owned\,\Gamma$$

$$free(p_1, q_1) \cap writes(c_2) = \{\}$$

$$free(p_2, q_2) \cap writes(c_1) = \{\}$$

critical variables must be protected!

cf. Owicki-Gries

region

$*$ instead of $\wedge$

$$\frac{\Gamma \vdash \{(p \star R) \wedge b\} c \{q \star R\}}{\Gamma, r(X){:}R \vdash \{p\} \textbf{with } r \textbf{ when } b \textbf{ do } c \{q\}}$$

if *R precise*

and $\mathit{free}(p, q) \cap X = \{\}$

$X \cap \mathit{owned}\,\Gamma = \{\}$

$X \cap \mathit{free}\,\Gamma = \{\}$

cf. Owicki-Gries

local resource

$*$ instead of $\wedge$

$$\frac{\Gamma, r(X){:}R \vdash \{p\}c\{q\}}{\Gamma \vdash \{p \star R\}\textbf{resource}\ r\ \textbf{in}\ c\{q \star R\}}$$

cf. Owicki-Gries

frame

$$\frac{\Gamma \vdash \{p\} c \{q\}}{\Gamma \vdash \{p \star R\} c \{q \star R\}}$$

$$\textit{if } \; free(R) \cap writes(c) = \{\}$$

$$\textit{and } \; free(R) \cap owned\,\Gamma = \{\}$$

IMPORTANT:
allows derivation of
non-tight properties

consequence

$$\frac{\Gamma \Leftrightarrow \Gamma' \qquad p \Rightarrow p' \qquad \Gamma' \vdash \{p'\} c \{q'\} \qquad q' \Rightarrow q}{\Gamma \vdash \{p\} c \{q\}}$$

where $\Gamma \Leftrightarrow \Gamma'$ means

same resource names,
same protection lists,
equivalent invariants

cf. Hoare logic, ...

concurrent disposal

$$\Gamma \vdash \{p\}\mathbf{dispose}\, x \parallel \mathbf{dispose}\, y\{q\}$$

provable iff

$$p \;\Rightarrow\; (x \mapsto \_) \star (y \mapsto \_) \star q$$

and *x*, *y* not in Γ

and free(*p*,*q*) ∩ owned(Γ) = { }

PUT and GET

$$\Gamma \;=\; buf(c, full) : (full \wedge c \mapsto \_) \vee (\neg full \wedge \mathbf{emp})$$

$$\Gamma \vdash \{x \mapsto \_\}\mathrm{PUT}\{\mathbf{emp}\}$$

$$\Gamma \vdash \{\mathbf{emp}\}\mathrm{GET}\{y \mapsto \_\}$$

$$\begin{aligned}\Gamma \vdash &\{\mathbf{emp}\} \\ &(x{:=}\mathbf{cons}(-); \mathrm{PUT}) \| (\mathrm{GET}; \mathbf{dispose}\, y) \\ &\{\mathbf{emp}\}\end{aligned}$$

all provable

PUT and GET

$$\Gamma' \;=\; \mathit{buf}(c, \mathit{full}) : (\mathit{full} \wedge \mathbf{emp}) \vee (\neg \mathit{full} \vee \mathbf{emp})$$

$$\Gamma' \vdash \{x \mapsto \_\}\mathrm{PUT}\{x \mapsto \_\}$$

$$\Gamma' \vdash \{\mathbf{emp}\}\mathrm{GET}\{\mathbf{emp}\}$$

$$\Gamma' \vdash \{\mathbf{emp}\} \quad (x{:=}\mathbf{cons}(-); \mathrm{PUT}; \mathbf{dispose}\, x) \| \mathrm{GET} \quad \{\mathbf{emp}\}$$

all provable

ownership

- Correctness proofs involve *ownership transfer*
  - protected variables
  - sub-heap determined by invariant
- A resource context specifies a *transfer policy*
- Logic ensures that processes *mind their own business*
  - operate on *separate* sub-heaps, ...
- To formalize this we introduce local state...

local state

- Process starts with *non-critical* data in its local state
- Local state *grows* when a resource is *acquired*
- Local state *shrinks* when a resource is *released*
- Error if process action breaks design rules

local state

$$(s, h, A)$$

- local store s : $\mathbf{Ide} \rightharpoonup V$
- local heap h : $\mathbf{Loc} \rightharpoonup V$
- resources A held by process

satisfying

$$dom\ s \cap owned\ \Gamma = owned(\Gamma \lceil A)$$

local store only contains protected variables for which the process has resources

local effect

$$(s, h, A) \xrightarrow[\Gamma]{\delta} (s, h, A)$$

$$(s, h, A) \xrightarrow[\Gamma]{i=v} (s, h, A) \qquad \mathit{if}\ i \in \mathit{dom}\, s$$

$$(s, h, A) \xrightarrow[\Gamma]{i:=v} ([s \mid i : v], h, A)$$

$$\mathit{if}\ i \in \mathit{dom}\, s - \mathit{free}(\Gamma \backslash A)$$

+ heap ops, as before

local effect

$$(s, h, A) \xrightarrow[\Gamma]{acq\, r} (s \cdot s', h \cdot h', A \cup \{r\})$$

$$\begin{array}{l} if\ \ r(X){:}R \in \Gamma \\ and\ \ s \perp s', h \perp h', dom\, s' = X, \\ (s \cdot s', h') \models R \end{array}$$

$$(s, h, A) \xrightarrow[\Gamma]{rel\, r} (s \backslash X, h - h', A - \{r\})$$

$$\begin{array}{l} if\ \ r(X){:}R \in \Gamma \\ h' \subseteq h,\ (s, h') \models R \end{array}$$

local effect

$$(s, h, A) \xrightarrow[\Gamma]{i:=v} \mathbf{abort}$$

$$\mathit{if}\;\; i \in \mathit{free}(\Gamma \backslash A) \;\; \mathit{or}\; i \notin \mathit{dom}\; s$$

$$(s, h, A) \xrightarrow[\Gamma]{\mathit{rel}\, r} \mathbf{abort}$$

$$\mathit{if}\;\; r(X){:}R \in \Gamma$$
$$\mathit{and}\; \forall h' \subseteq h.\; (s, h') \models \neg R$$

+ read, heap ops, as before

... breaking the design rules

local computation

- What a *process* sees of an interactive computation
- Assumes that the *environment*
  - respects the resource rules
  - interferes only on synchronization

$$(s, h, A) \xrightarrow[\Gamma]{\alpha} (s', h', A')$$

$$(s, h, A) \xrightarrow[\Gamma]{\alpha} \mathbf{abort}$$

defined by composition

A local computation

of PUT || (GET; dispose y)

$$\Gamma \;=\; \mathit{buf}(c, \mathit{full}) : (\mathit{full} \wedge c \mapsto \_) \vee (\neg \mathit{full} \wedge \mathbf{emp})$$

$$([x : v, y : \_], [v : \_], \{\})$$

$$\xrightarrow[\Gamma]{\mathit{acq}\ \mathit{buf}} ([x : v, y : \_, \mathit{full} : \mathit{false}, c : \_], [v : \_], \{\mathit{buf}\})$$

$$\xrightarrow[\Gamma]{\mathit{full}=\mathit{false}\ \mathrm{put}\ v} ([x : v, y : \_, \mathit{full} : \mathit{true}, c : v], [v : \_], \{\mathit{buf}\})$$

$$\xrightarrow[\Gamma]{\mathit{rel}\ \mathit{buf}} ([x : v, y : \_], [\,], \{\})$$

$$\xrightarrow[\Gamma]{\mathit{acq}\ \mathit{buf}} ([x : v, y : \_, \mathit{full} : \mathit{true}, c : v], [v : \_], \{\mathit{buf}\})$$

$$\xrightarrow[\Gamma]{\mathit{full}=\mathit{true}\ \mathrm{get}\ v} ([x : v, y : v, \mathit{full} : \mathit{false}, c : v], [v : \_], \{\mathit{buf}\})$$

$$\xrightarrow[\Gamma]{\mathit{rel}\ \mathit{buf}} ([x : v, y : v], [v : \_], \{\})$$

$$\xrightarrow[\Gamma]{y=v\ \mathrm{disp}\ v} ([x : v, y : v], [\,], \{\})$$

A local computation

of PUT

$$\Gamma \;=\; buf(c, full) : (full \wedge c \mapsto \_) \vee (\neg full \wedge \mathbf{emp})$$

$$([x : v], [v : \_], \{\})$$

$$\xrightarrow[\Gamma]{acq\; buf} \quad ([x : v, full : false, c : \_], [v : \_], \{buf\})$$

$$\xrightarrow[\Gamma]{full = false\; \mathrm{put}\; v} \quad ([x : v, full : true, c : v], [v : \_], \{buf\})$$

$$\xrightarrow[\Gamma]{rel\; buf} \quad ([x : v], [\;], \{\})$$

A local computation

of GET; dispose y

$$\Gamma \;=\; \mathit{buf}(c, \mathit{full}) : (\mathit{full} \wedge c \mapsto \_) \vee (\neg \mathit{full} \wedge \mathbf{emp})$$

$$([y:\_], [\,], \{\})$$

$$\xrightarrow[\Gamma]{\mathit{acq}\ \mathit{buf}} ([y:\_, \mathit{full}:\mathit{true}, c:v], [v:\_], \{\mathit{buf}\})$$

$$\xrightarrow[\Gamma]{\mathit{full}=\mathit{true}\ \mathrm{get}\ v} ([y:v, \mathit{full}:\mathit{false}, c:v], [v:\_], \{\mathit{buf}\})$$

$$\xrightarrow[\Gamma]{\mathit{rel}\ \mathit{buf}} ([y:v], [v:\_], \{\})$$

$$\xrightarrow[\Gamma]{y=v\ \mathrm{disp}\ v} ([y:v], [\,], \{\})$$

Validity

$$\Gamma \vdash \{p\}c\{q\}$$

Every finite *local computation* of c

from a *local state* satisfying p

is error-free

and

ends in a local state satisfying q

$$\forall \alpha \in [\![c]\!].$$
$$\forall s : dom\, s \supseteq free(c) - owned\, \Gamma.$$
$$(s, h) \models p \;\&\; (s, h) \xrightarrow[\Gamma]{\alpha} \sigma' \;\Rightarrow\; \sigma' \models q$$

Soundness

THEOREM

- Every provable formula is valid

PROOF

- uses local states, local effects
- show that each rule preserves validity
- for PARALLEL rule use Parallel Lemma

Parallel Lemma

- A local computation of $c_1 || c_2$ decomposes into local computations of c_1 and c_2
- A local error of $c_1 || c_2$ is caused by a local error of c_1 or c_2 (not by interference)
- A successful local computation of $c_1 || c_2$ is consistent with all successful local computations of c_1 and c_2

Parallel Lemma

Suppose

$$free(c_1) \cap writes(c_2) \subseteq owned\,\Gamma$$
$$free(c_2) \cap writes(c_1) \subseteq owned\,\Gamma$$
$$\alpha_1 \in [\![c_1]\!],\ \alpha_2 \in [\![c_2]\!],\ \alpha \in \alpha_1 \| \alpha_2,\ h = h_1 \cdot h_2$$

If

$$(s, h) \xrightarrow[\Gamma]{\alpha} \mathbf{abort}$$

then

$$(s \backslash writes(c_2), h_1) \xrightarrow[\Gamma]{\alpha_1} \mathbf{abort}$$

or

$$(s \backslash writes(c_1), h_2) \xrightarrow[\Gamma]{\alpha_2} \mathbf{abort}$$

Parallel Lemma

Suppose

$$free(c_1) \cap writes(c_2) \subseteq owned\,\Gamma$$

$$free(c_2) \cap writes(c_1) \subseteq owned\,\Gamma$$

$$\alpha_1 \in [\![c_1]\!],\ \alpha_2 \in [\![c_2]\!],\ \alpha \in \alpha_1 \| \alpha_2,\ h = h_1 \cdot h_2$$

If

$$(s, h) \xrightarrow[\Gamma]{\alpha} (s', h')$$

$$(s \backslash writes(c_2), h_1) \xrightarrow[\Gamma]{\alpha_1} (s'_1, h'_1)$$

$$(s \backslash writes(c_1), h_2) \xrightarrow[\Gamma]{\alpha_2} (s'_2, h'_2)$$

then

$$s'_1 = s' \backslash writes(c_2)$$

$$s'_2 = s' \backslash writes(c_1)$$

$$h' = h'_1 \cdot h'_2$$

Local vs. **global**

- Soundness shows that *provable* formulas are *valid*
- *Validity* refers to *local* computations
- Need to connect with conventional notions
  - *global* state
  - traditional partial correctness

... local computations
are consistent with global view...

Connection Lemma

Suppose $\alpha \in [\![c]\!],\ h = h_1 \cdot h_2,\ (s, h_2) \models inv(\Gamma)$

If

$$(s, h) \stackrel{\alpha}{\Longrightarrow} \mathbf{abort}$$

then

$$(s \backslash owned\,\Gamma, h_1) \xrightarrow[\Gamma]{\alpha} \mathbf{abort}$$

If

$$(s, h) \stackrel{\alpha}{\Longrightarrow} (s', h')$$

then

$$(s \backslash owned\,\Gamma, h_1) \xrightarrow[\Gamma]{\alpha} (s'_1, h'_1)$$

$$s'_1 = s' \backslash owned\,\Gamma$$

$$\exists h'_2.\ h' = h'_1 \cdot h'_2\ \&\ (s', h'_2) \models inv(\Gamma)$$

Corollary

Validity implies error-freedom

$$\Gamma \vdash \{p\}c\{q\}$$

Every finite *computation* of c
from a global state satisfying

$$p \star inv(\Gamma)$$

is error-free,
and ends in a state satisfying

$$q \star inv(\Gamma)$$

cf. traditional notion of validity

Conclusions

- Concurrent separation logic extends and generalizes Owicki-Gries, Hoare
- Supports *local reasoning*
  - important for scalability
- Suitable for wide variety of programs

parallel sorting *semaphores*

garbage collection *readers/writers*

Further topics

- Simple recursive procedures

  an obvious extension *cf. Reynolds*

- More general logics

  permissions *Bornat, Calcagno, O'Hearn, Parkinson*

- Automation

  Smallfoot *Berdine, Calcagno, O'Hearn*