A semantics for concurrent separation logic

Stephen Brookes
Carnegie Mellon University

THANKS

- Peter O'Hearn
- John Reynolds
- Josh Berdine

LANGUAGE

- Concurrent programs
  - shared mutable data
- Resources
  - mutual exclusion
- Synchronization
  - conditional critical regions

Hoare, Owicki/Gries

Race conditions

cause unpredictable behavior

- Concurrent write to shared variable
- Concurrent update/disposal of heap cell

Race conditions

cause unpredictable behavior

- Concurrent write to shared variable
- Concurrent update/disposal of heap cell

Race conditions

cause unpredictable behavior

- Concurrent write to shared variable
- Concurrent update/disposal of heap cell

SEMANTIC FRAMEWORK

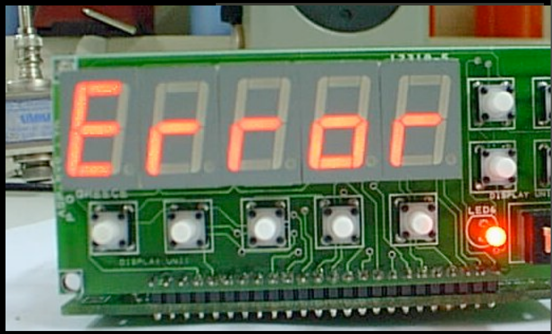
- A process denotes a set of *action traces*
  - *actions* have *effect* on *state*
- Traces describe *interactive computations*
  - fair interleaving
  - mutually exclusive resources
  - treat *race condition* as *disaster*

SEMANTIC FRAMEWORK

- A process denotes a set of *action traces*
  - *actions* have *effect* on *state*
- Traces describe *interactive computations*
  - fair interleaving
  - mutually exclusive resources
  - treat *race condition* as *disaster*

SEMANTIC FRAMEWORK

- A process denotes a set of *action traces*
  - *actions* have *effect* on *state*
- Traces describe *interactive computations*
  - fair interleaving
  - mutually exclusive resources
  - treat *race condition* as *disaster*



ISSUES

concurrency + pointers

- Race conditions
  - unpredictable behavior
  - not statically detectable
- Partial and total correctness
- Safety and liveness properties
- Deadlock

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

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

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

TRACES

- sequences 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

CONCATENATION

$$T_1 T_2 = \{\alpha_1 \alpha_2 \mid \alpha_1 \in T_1 \ \& \ \alpha_2 \in T_2\}$$

ITERATION

$$T^* = \bigcup_{n=0}^{\infty} T^n$$

$$T^\infty = T^* \cup T^\omega$$

TRACE SEMANTICS

- Integer expressions

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

- Boolean expressions

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

- List expressions

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

- Commands

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

TRACE SEMANTICS

- Integer expressions

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

- Boolean expressions

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

- List expressions

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

- Commands

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

defined denotationally

SEMANTIC DEFINITIONS

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

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

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

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

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

SEMANTIC DEFINITIONS

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

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

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

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

REGION

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

where

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

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

LOCAL VARIABLES

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

α sequential for i

... i=v ... i=v i:=v' ... i=v' ...

local actions hidden in $\alpha\backslash i$

i=0 x=1 i:=2 x:=2

⬇

x=1 x:=2

SEQUENTIAL TRACES

assume no interference

- sequential for *i*

  each read yields the most recently written value

- sequential for *r*

  available at start
  acquire before release

LOCAL RESOURCE

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

α sequential for r

local actions hidden in $\alpha \backslash r$

PARALLEL

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

- each process starts with no resources
- resources are mutually exclusive
- a race is an error

mutex fairmerge
with race detection

Parallel composition

- what each process can do depends on its resources and those of its environment

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

- a race is interpreted as an error

$$\lambda_1 \bowtie \lambda_2$$

Mutex constraint

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

Definition

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

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

$$(A_1, A_2) \xrightarrow{\lambda} (A_1, A_2) \qquad \lambda \neq acq(r), rel(r)$$

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

Interfering actions

Definition

$$\lambda_1 \bowtie \lambda_2$$

iff

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

or

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

concurrent read/write
concurrent write/write
to store or heap

Mutex fairmerge

$$\alpha_{1\,A_1}\|_{A_2}\alpha_2$$

Inductive definition

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

$$(\lambda_1\alpha_1)_{A_1}\|_{A_2}(\lambda_2\alpha_2) = \{abort\} \qquad \text{if } \lambda_1 \bowtie \lambda_2$$

$$\begin{aligned}
&= \{\lambda_1\beta \mid (A_1, A_2) \xrightarrow{\lambda_1} (A_1', A_2) \;\&\; \beta \in \alpha_1 \;{}_{A_1'}\|_{A_2}\, (\lambda_2\alpha_2)\} \\
&\cup \{\lambda_2\beta \mid (A_2, A_1) \xrightarrow{\lambda_2} (A_2', A_1) \;\&\; \beta \in (\lambda_1\alpha_1) \;{}_{A_1}\|_{A_2'}\, \alpha_2\} \\
&\qquad \text{otherwise}
\end{aligned}$$

Mutex fairmerge

$$\alpha_{1\,A_1}\|_{A_2}\alpha_2$$

Inductive definition

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

$$(\lambda_1\alpha_1)_{A_1}\|_{A_2}(\lambda_2\alpha_2) \ = \ \{abort\} \qquad \text{if } \lambda_1 \bowtie \lambda_2$$

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

Mutex fairmerge

$$\alpha_1 {}_{A_1}\|_{A_2} \alpha_2$$

Inductive definition

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

$$(\lambda_1\alpha_1)_{A_1}\|_{A_2}(\lambda_2\alpha_2) \;=\; \{\mathit{abort}\} \qquad \text{if } \lambda_1 \bowtie \lambda_2$$

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

$$\text{otherwise}$$

READS & WRITES

$$writes([v]{=}v') = \{\}$$
$$writes([v]{:=}v') = \{v\}$$
$$writes(alloc(v, [v_0, \ldots, v_n])) = \{v, \ldots, v+n\}$$
$$writes(disp(v)) = \{v\}$$
$$writes(i{=}v) = \{\}$$
$$writes(i{:=}v) = \{i\}$$

$$free([v]{=}v') = \{v\}$$
$$free([v]{:=}v') = \{v\}$$
$$free(alloc(v, [v_0, \ldots, v_n])) = \{v, \ldots, v+n\}$$
$$free(disp(v)) = \{v\}$$
$$free(i{=}v) = \{i\}$$
$$free(i{:=}v) = \{i\}$$

EXAMPLES

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

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

$$[\![x{:=}x+1 \| x{:=}x+1]\!] = \{x{=}v\; abort \mid v \in V\}$$

PUT

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

Typical trace

$$acq(buf)\ full{=}\textbf{false}\ put(v)\ rel(buf)$$

where

$$put(v) =_{\text{def}} x{=}v\ c{:=}v\ full{=}\textbf{true}$$

GET

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

Typical trace

$$\mathit{acq}(\mathit{buf})\ \mathit{full}{=}\textbf{true}\ \mathit{get}(v')\ \mathit{rel}(\mathit{buf})$$

where

$$\mathit{get}(v') =_{\mathrm{def}} c{=}v'\ y{:=}v'\ \mathit{full}{:=}\textbf{false}$$

DEADLOCK

$$\begin{aligned}&\textbf{resource } r_1, r_2 \textbf{ in}\\ &\qquad \textbf{with } r_1 \textbf{ do with } r_2 \textbf{ do } x{:=}1\\ &\quad \parallel \textbf{with } r_2 \textbf{ do with } r_1 \textbf{ do } y{:=}1\end{aligned}$$

has traces

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

GLOBAL STATE

(s, h, A)

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

GLOBAL STATE

(s, h, A)

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

(s, h) when resource set is empty

EFFECTS

- State may *enable* action
- Enabled action causes *state change* or *error*

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

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

EFFECTS

- State may *enable* action
- Enabled action causes *state change* or *error*

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

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

defined by cases

EFFECTS

of store actions

Definition

$$(s, h, A) \xrightarrow{i=v} (s, h, A)$$

$$\text{if } (i, v) \in s$$

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

$$\text{if } (i, v) \in dom(s)$$

EFFECTS

of heap actions

Definition

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

$$(s, h, A) \xrightarrow{[v]:=v'} (s, [h \mid v : v'], A) \quad \text{if } v \in \mathtt{dom}(h)$$

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

$$(s, h, A) \xrightarrow{disp(v)} (s, h \backslash v, A) \quad \text{if } v \in \mathtt{dom}(h)$$

EFFECTS

of resource actions

Definition

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

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

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

EFFECTS

causing error

Definition

$$(s, h, A) \xrightarrow{i=v} \textbf{abort} \quad \text{if } i \notin \mathtt{dom}(s)$$

$$(s, h, A) \xrightarrow{i:=v} \textbf{abort} \quad \text{if } i \notin \mathtt{dom}(s)$$

$$(s, h, A) \xrightarrow{[v]=v'} \textbf{abort} \quad \text{if } v \notin \mathtt{dom}(h)$$

$$(s, h, A) \xrightarrow{[v]:=v'} \textbf{abort} \quad \text{if } v \notin \mathtt{dom}(h)$$

EFFECTS

Definition

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

$$\mathbf{abort} \xrightarrow{\lambda} \mathbf{abort}$$

GLOBAL COMPUTATION

- Consecutive sequence of actions
- A *sequential* trace
  - no interference between steps

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

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

GLOBAL COMPUTATION

- Consecutive sequence of actions
- A *sequential* trace
  - 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 global computation

of PUT || (GET; dispose y)

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

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

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

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

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

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

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

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

ERROR-FREE

Definition

c is error-free from (s, h)

if

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

ERROR-FREE

Definition

c is error-free from (s, h)

if

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

EXAMPLE

$$\mathbf{dispose}(x) \parallel \mathbf{dispose}(y)$$

is error-free iff

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

ERROR-FREE?

- PUT || (GET; dispose y)
  if $s(full) = true \;\&\; s(c) \in dom(h)$

  or $s(full) = false \;\&\; s(x) \in dom(h)$
- (PUT; dispose x) || GET
  if $s(full) \in \{true, false\} \;\&\; s(x) \in dom(h)$
- (PUT; dispose x) || (GET; dispose y)
  never

ERROR-FREE?

PUT || (GET; dispose y)

if $s(full) = true \;\&\; s(c) \in dom(h)$

or $s(full) = false \;\&\; s(x) \in dom(h)$

- (PUT; dispose x) || GET

  if $s(full) \in \{true, false\} \;\&\; s(x) \in dom(h)$

- (PUT; dispose x) || (GET; dispose y)

  never

ERROR-FREE?

PUT || (GET; dispose y)

if $s(full) = true$ & $s(c) \in dom(h)$

or $s(full) = false$ & $s(x) \in dom(h)$

(PUT; dispose x) || GET

if $s(full) \in \{true, false\}$ & $s(x) \in dom(h)$

- (PUT; dispose x) || (GET; dispose y)

  never

ERROR-FREE?

PUT || (GET; dispose y)

if $s(full) = true$ & $s(c) \in dom(h)$

or $s(full) = false$ & $s(x) \in dom(h)$

(PUT; dispose x) || GET

if $s(full) \in \{true, false\}$ & $s(x) \in dom(h)$

(PUT; dispose x) || (GET; dispose y)

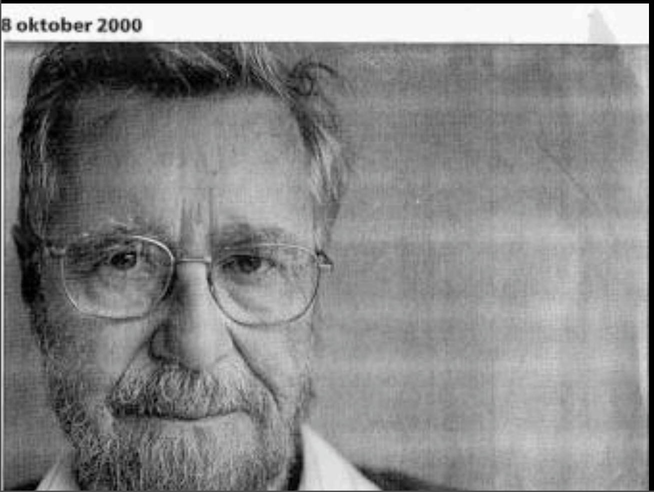
never

So far...

- Trace semantics
  - compositional
  - allows race detection
- Hard to use directly by itself...
  - doesn't reflect "loosely connected" principle

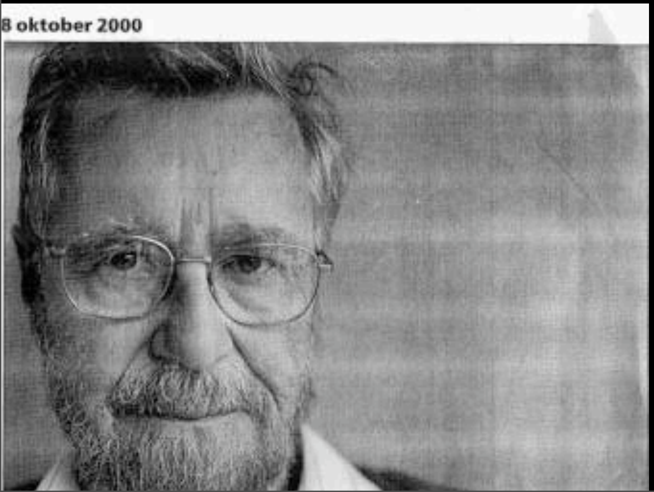
So far...

- Trace semantics
  - compositional
  - allows race detection
- Hard to use directly by itself...
  - doesn't reflect "loosely connected" principle



So far...

- Trace semantics
  - compositional
  - allows race detection
- Hard to use directly by itself...
  - doesn't reflect "loosely connected" principle



well-designed programs
should be easier
to prove correct...

CONCURRENT SEPARATION LOGIC

- Resource-sensitive partial correctness
  - for *loosely coupled* processes
  - proof rules guarantee race-freedom

building on ideas of
Dijkstra,
Hoare, Owicki/Gries,
Reynolds, O'Hearn

Owicki/Gries/Hoare

- Partition the *critical identifiers*
  - among processes, available resources
- Maintain *conjunction* of resource invariants
  - for the available resources
- Rules enforce this discipline
  - rely/guarantee at synchronization points
  - scoped protection of critical storage

Owicki/Gries/Hoare

NOT SOUND FOR POINTER PROGRAMS

- Partition the *critical identifiers*
  - among processes, available resources
- Maintain *conjunction* of resource invariants
  - for the available resources
- Rules enforce this discipline
  - rely/guarantee at synchronization points
  - scoped protection of critical storage

O'Hearn

- Partition the *critical identifiers* *and the heap*
  - among processes, available resources
- Maintain *separate conjunction* of resource invariants
  - for available resources, *in sub-heap*
- Rules enforce this discipline
  - rely/guarantee at synchronization points
  - *dynamic* transfer of critical storage *and heap*

DESIGN RULES

- Associate resources with *protection lists*
  - *critical* variables must be protected
  - *protected* data must be accessed inside region
- Hide details with *resource invariant*

Hoare, Owicki/Gries

- Use *separation logic*

O'Hearn, Reynolds

RESOURCE CONTEXTS

$$\Gamma ::= r_1(X_1){:}R_1, \ldots, r_k(X_k){:}R_k$$

satisfying *modularity properties*

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

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

- resource names $dom(\Gamma) = \{r_1, \ldots, r_k\}$
- protection lists $owned(\Gamma) = X_1 \cup \cdots \cup X_k$
- invariants $inv(\Gamma) = R_1 * \cdots * R_k$

PRECISION

We assume each invariant is 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 precise resource invariant uniquely determines a heap portion...

SPECIFICATIONS

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

Well-formed when

- critical identifiers of c are protected
- protected identifiers only accessed in region
- free identifiers of invariants only changed in region
- p and q don't mention protected data

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

Properties enforced by the inference rules

PARALLEL RULE

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

if

$$\begin{aligned} &free(c_1) \cap writes(c_2) \subseteq owned(\Gamma) \\ &free(c_2) \cap writes(c_1) \subseteq owned(\Gamma) \\ &free(p_2, q_2) \cap writes(c_1) = \{\} \\ &free(p_1, q_1) \cap writes(c_2) = \{\} \end{aligned}$$

REGION RULE

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

if *R precise*

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

$X \cap \mathit{owned}(\Gamma) = \{\}$

$X \cap \mathit{free}(\Gamma) = \{\}$

RESOURCE RULE

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

VALIDITY?

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

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

$$p * inv(\Gamma)$$

is error-free,
and ends in a state satisfying

$$q * inv(\Gamma)$$

VALIDITY?

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

Every finite *computation* of c

from a global state satisfying

$$p * inv(\Gamma)$$

is error-free,

and ends in a state satisfying

$$q * inv(\Gamma)$$

NOT COMPOSITIONAL

VALIDITY

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

Every finite *interactive computation* of c
in an environment that respects Γ
from a global state satisfying
$p * inv(\Gamma)$
is error-free, respects Γ,
and ends in a state satisfying
$q * inv(\Gamma)$

An informal working definition for now...

INFERENCE RULES

based on

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

$\wedge \mapsto *$

a simple trick
with deep ramifications

SKIP

$$\overline{\Gamma \vdash \{p\}\mathbf{skip}\{p\}}$$

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

ASSIGNMENT

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

$$\text{if } i \notin \mathit{owned}(\Gamma) \cup \mathit{free}(\Gamma)$$

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

LOOKUP

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

if $i \notin \mathit{free}(\Gamma) \cup \mathit{owned}(\Gamma)$

and $i \notin \mathit{free}(e, e')$

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

UPDATE

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

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

ALLOCATE

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

$$\text{if } i \notin \mathit{free}(E)$$

$$\text{and } \mathit{free}(E) \cap \mathit{owned}(\Gamma) = \{\}$$

$$\text{and } i \notin \mathit{free}(\Gamma) \cup \mathit{owned}(\Gamma)$$

DISPOSE

$$\frac{}{\Gamma \vdash \{e \mapsto -\}\,\mathbf{dispose}(e)\,\{\mathbf{emp}\}}$$

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

RENAMING

$$\frac{\Gamma \vdash \{p\}\textbf{resource } r' \textbf{ in } [r'/r]c\{q\}}{\Gamma \vdash \{p\}\textbf{resource } r \textbf{ in } c\{q\}}$$

$$\text{if } r' \notin \texttt{res}(c)$$

AUXILIARY

$$\frac{\Gamma \vdash \{p\}c\{q\}}{\Gamma \vdash \{p\}c\backslash X\{q\}}$$

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

$X\ auxiliary\ for\ c$

AUXILIARY VARIABLES

- A set X is auxiliary for c if each free occurrence in c of an identifier from X is in an assignment whose target is in X
  - no effect on control flow
  - no effect on other variables

FRAME

$$\frac{\Gamma \vdash \{p\}c\{q\}}{\Gamma \vdash \{p * I\}c\{q * I\}}$$

if $\quad free(I) \cap writes(c) = \{\}$

$\quad free(I) \cap owned(\Gamma) = \{\}$

EXPANSION

$$\frac{\Gamma \vdash \{p\}c\{q\}}{\Gamma, \Gamma' \vdash \{p\}c\{q\}}$$

if Γ, Γ' disjoint

and $\mathit{free}(p, c, q) \cap \mathit{owned}(\Gamma') = \{\}$

and $\mathit{writes}(c) \cap \mathit{free}(\Gamma') = \{\}$

CONTRACTION

$$\frac{\Gamma,\Gamma' \vdash \{p\}c\{q\}}{\Gamma \vdash \{p\}c\{q\}}$$

$$\text{if } \Gamma, \Gamma' \text{ disjoint}$$

$$\text{and } res(c) \subseteq dom(\Gamma)$$

CONSEQUENCE

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

CONCURRENT DISPOSAL

$$\Gamma \vdash \{p\}\mathbf{dispose}(x)\|\mathbf{dispose}(y)\{q\}$$

valid if

$$p \Rightarrow (x \mapsto -) * (y \mapsto -) * q$$

PUT and GET

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

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

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

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

valid formulas

OWNERSHIP

- Correctness proofs involve dynamic transfer
  - heap associated with resources
    - determined by invariants
- Key concept underpinning soundness proof
  - must show that transfer policy is safe
- Difficult to manage using *global state*
- Solution: local state, local computations

KEY IDEAS

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

LOCAL STATES

(s, h, A)

- Local visibility

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

- An action has a local effect

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

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

LOCAL EFFECTS

on local states

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

$$(s, h, A) \xrightarrow[\Gamma]{i:=v} ([s \mid i : v], h, A)$$
$$\text{if } i \in \mathtt{dom}(s) - \mathit{free}(\Gamma \backslash A)$$

LOCAL EFFECTS

following the design rules

$$(s, h, A) \xrightarrow[\Gamma]{acq(r)} (s \cdot s', h \cdot h', A \cup \{r\})$$
$$\begin{array}{ll} \text{if} & r(X) : R \in \Gamma \\ \text{and} & s \perp s', h \perp h', \mathtt{dom}(s') = X, \\ & (s \cdot s', h') \models R \end{array}$$

$$(s, h, A) \xrightarrow[\Gamma]{rel(r)} (s \backslash X, h - h', A - \{r\})$$
$$\begin{array}{ll} \text{if} & r(X) : R \in \Gamma \\ \text{and} & h' \subseteq h, (s, h') \models R \end{array}$$

LOCAL ERRORS

breaking the design rules

$$(s, h, A) \xrightarrow[\Gamma]{i:=v} \textbf{abort} \quad \text{if } i \in \mathit{free}(\Gamma \backslash A) \text{ or } i \notin \mathit{dom}(s)$$

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

$$\text{if } \quad r(X) : R \in \Gamma$$

$$\text{and } \quad \forall h' \subseteq h.\ (s, h') \models \neg R$$

LOCAL COMPUTATION

- What a *process* sees of an interactive computation in an environment that respects the resource context
- Interference 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)

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

$$
\begin{array}{rl}
& ([x : v, y : \_], [v : \_], \{\}) \\
\xrightarrow[\Gamma]{acq(buf)} & ([x : v, y : \_, \mathit{full} : \mathbf{false}, c : \_], [v : \_], \{\mathit{buf}\}) \\
\xrightarrow[\Gamma]{\mathit{full}=\mathbf{false}\ put(v)} & ([x : v, y : \_, \mathit{full} : \mathbf{true}, c : v], [v : \_], \{\mathit{buf}\}) \\
\xrightarrow[\Gamma]{rel(buf)} & ([x : v, y : \_], [\,], \{\}) \\
\xrightarrow[\Gamma]{acq(buf)} & ([x : v, y : \_, \mathit{full} : \mathbf{true}, c : v], [v : \_], \{\mathit{buf}\}) \\
\xrightarrow[\Gamma]{\mathit{full}=\mathbf{true}\ get(v)} & ([x : v, y : v, \mathit{full} : \mathbf{false}, c : v], [v : \_], \{\mathit{buf}\}) \\
\xrightarrow[\Gamma]{rel(buf)} & ([x : v, y : v], [v : \_], \{\}) \\
\xrightarrow[\Gamma]{y=v\ disp(v)} & ([x : v, y : v], [\,], \{\})
\end{array}
$$

A local computation

of PUT

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

$$
\begin{aligned}
& ([x:v],[v:\_],\{\}) \\
\xrightarrow[\Gamma]{acq(buf)} \; & ([x:v, full:\mathbf{false}, c:\_],[v:\_],\{buf\}) \\
\xrightarrow[\Gamma]{full=\mathbf{false}\; put(v)} \; & ([x:v, full:\mathbf{true}, c:v],[v:\_],\{buf\}) \\
\xrightarrow[\Gamma]{rel(buf)} \; & ([x:v],[\;],\{\})
\end{aligned}
$$

A local computation

of GET; dispose y

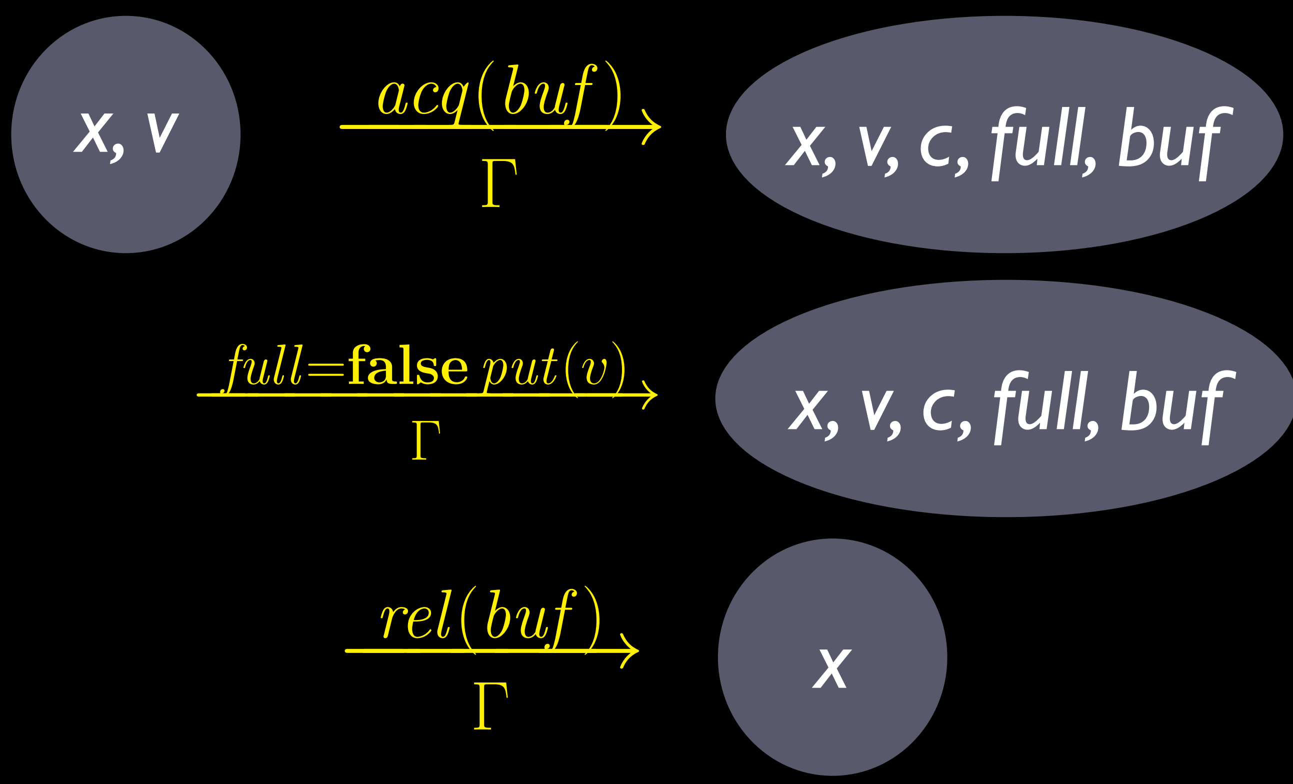
$$\Gamma \;=\; buf(c, full) : (full \wedge c \mapsto \_) \vee (\neg full \wedge \mathbf{emp})$$

$$
\begin{array}{l}
([y : \_], [\,], \{\}) \\
\xrightarrow[\Gamma]{acq(buf)} ([y : \_, full : \mathbf{true}, c : v], [v : \_], \{buf\}) \\
\xrightarrow[\Gamma]{full=\mathbf{true}\; get(v)} ([y : v, full : \mathbf{false}, c : v], [v : \_], \{buf\}) \\
\xrightarrow[\Gamma]{rel(buf)} ([y : v], [v : \_], \{\}) \\
\xrightarrow[\Gamma]{y=v\; disp(v)} ([y : v], [\,], \{\})
\end{array}
$$

A local computation

of PUT

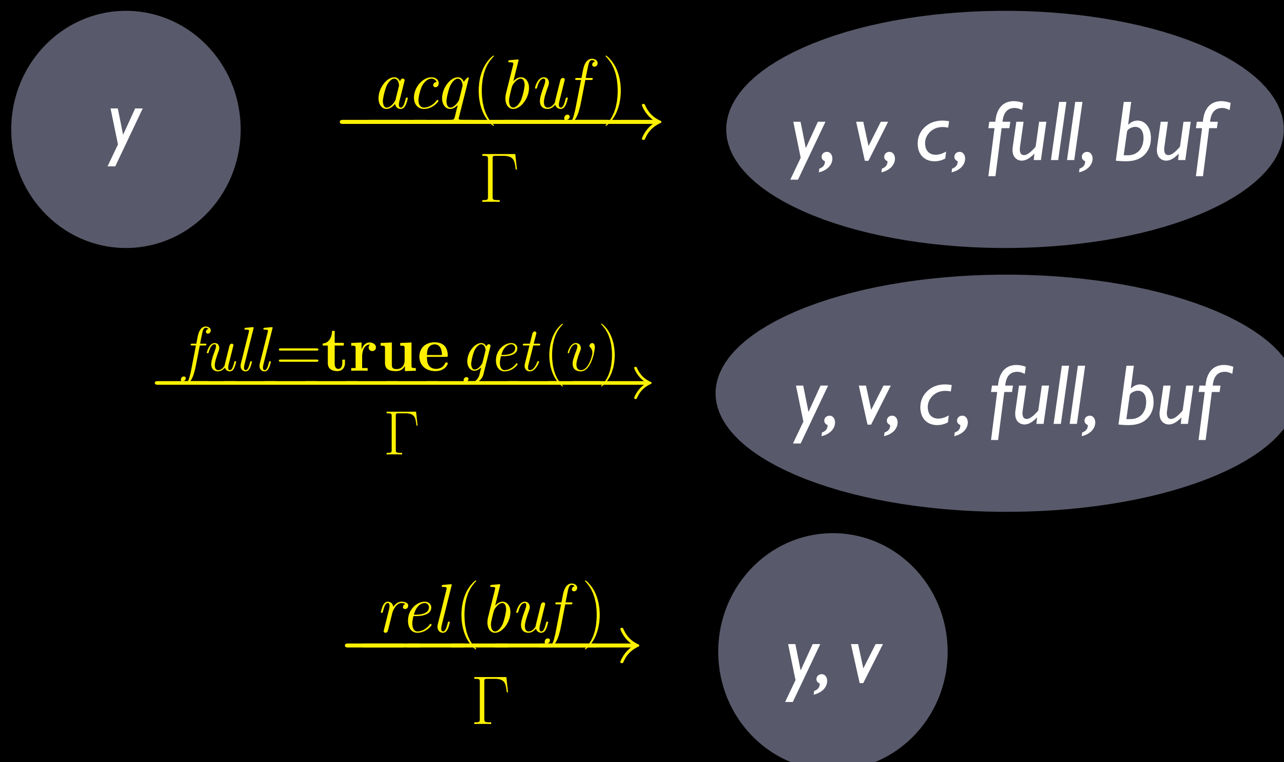
$$\Gamma \;=\; buf(c, full) : (full \wedge c \mapsto \_) \vee (\neg full \wedge \mathbf{emp})$$



A local computation

of GET

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



A local computation

of (PUT(x); dispose x) || GET(y)

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

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

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

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

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

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

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

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

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

Local computation

of (PUT(x); dispose x)

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

$$
\begin{aligned}
& ([x : v], [v : \_], \{\}) \\
\xrightarrow[\Gamma']{acq(buf)} & ([x : v, full : \mathbf{false}, c : \_], [v : \_], \{buf\}) \\
\xrightarrow[\Gamma']{full=\mathbf{false}\ put(v)} & ([x : v, full : \mathbf{true}, c : v], [v : \_], \{buf\}) \\
\xrightarrow[\Gamma']{rel(buf)} & ([x : v], [v : \_], \{\}) \\
\xrightarrow[\Gamma']{x=v\ disp(v)} & ([x : v], [\ ], \{\})
\end{aligned}
$$

Local computation

of GET(y)

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

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

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

$$\xrightarrow[\Gamma']{full=\mathbf{true}\; get(v)} ([y : v, full : \mathbf{false}, c : v], [\,], \{buf\})$$

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

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 state satisfying q

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

LOCALIZATION

$$(s, h, A) \Rightarrow (s{\downarrow}A, h, A)$$

global → local

$$\begin{aligned} s{\downarrow}A &= s \backslash owned(\Gamma) \cup s\lceil owned(\Gamma\lceil A) \\ &= s \backslash owned(\Gamma \backslash A) \end{aligned}$$

Special cases

$$s{\downarrow}\{\} \;=\; s \backslash owned(\Gamma)$$

$$s{\downarrow}\mathtt{dom}(\Gamma) \;=\; s$$

SOUNDNESS

- Every provable formula is valid
  - proof uses local states, local effects
  - show that each rule preserves validity
  - for PARALLEL rule use Parallel Lemma

FRAME LEMMA

Suppose $h = h_1 \cdot h_2, A = A_1 \cdot A_2$

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

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

and $\neg\ (s{\downarrow}A_1, h_1, A_1) \xrightarrow[\Gamma]{\lambda} \mathbf{abort}$

then $(s{\downarrow}A_1, h_1, A_1) \xrightarrow[\Gamma]{\lambda} (s'{\downarrow}A_1', h_1', A_1')$

$h' = h_1' \cdot h_2, A' = A_1' \cdot A_2$

PARALLEL LEMMA

When c_1 and c_2 are loosely coupled...

- 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 any successful local computations of c_1 and c_2

Loosely connected processes mind their own business

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

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

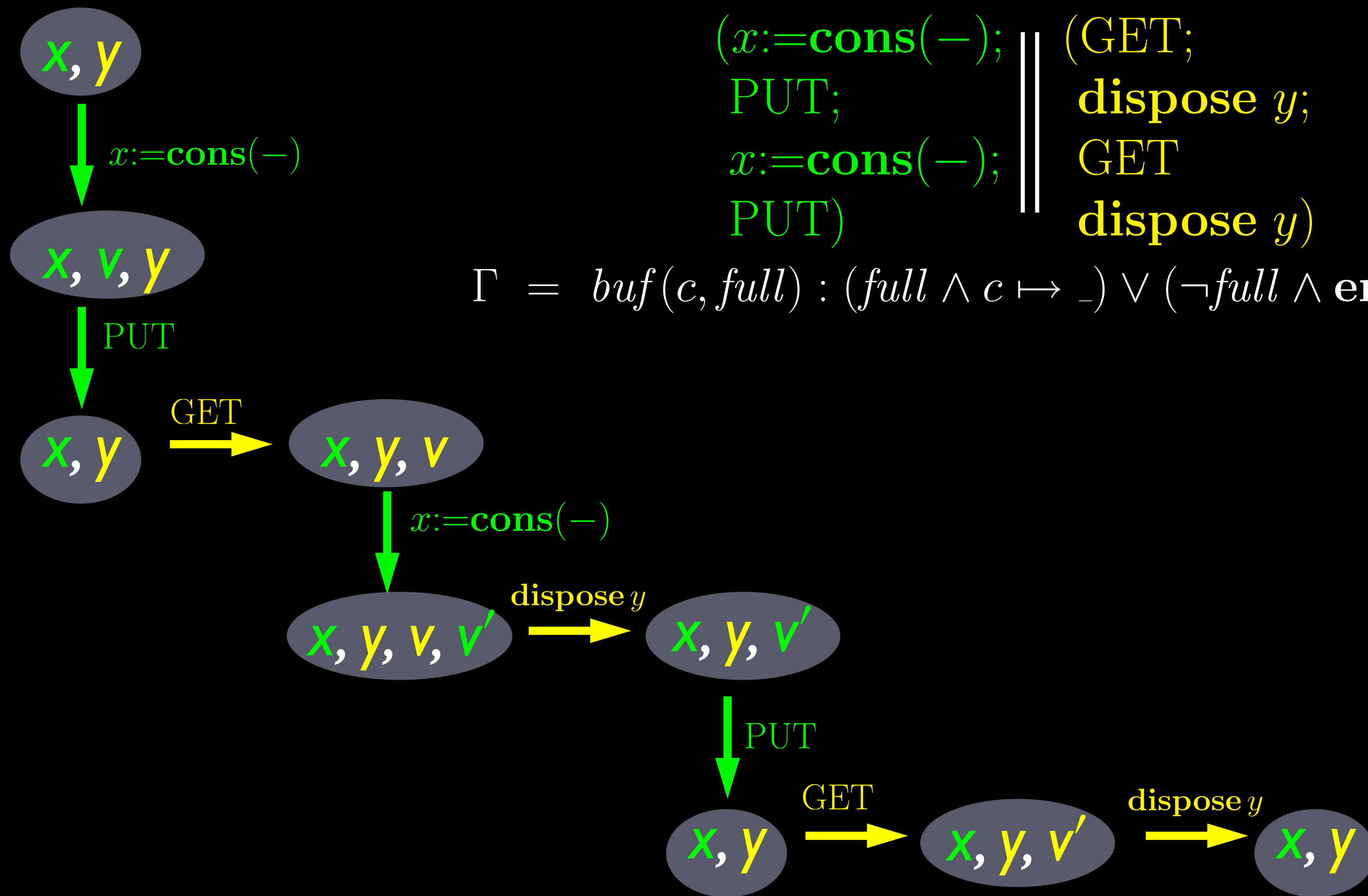
$$s'_2 = s' \backslash writes(c_1)$$

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

A LOCAL PARALLEL COMPUTATION

$$\begin{array}{l} (x{:=}\mathbf{cons}(-);\\ \ \mathrm{PUT};\\ \ x{:=}\mathbf{cons}(-);\\ \ \mathrm{PUT}) \end{array} \;\Big\|\; \begin{array}{l} (\mathrm{GET};\\ \ \mathbf{dispose}\ y;\\ \ \mathrm{GET}\\ \ \mathbf{dispose}\ y) \end{array}$$

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

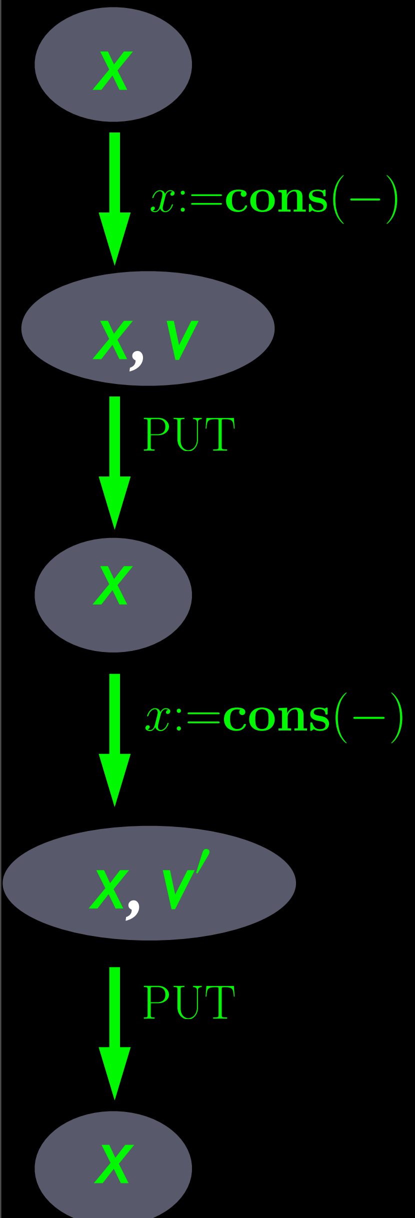


A LOCAL PARALLEL COMPUTATION

$$\begin{array}{l} (x{:=}\mathbf{cons}(-); \\ \ \mathrm{PUT}; \\ \ x{:=}\mathbf{cons}(-); \\ \ \mathrm{PUT}) \end{array} \Bigg\| \begin{array}{l} (\mathrm{GET}; \\ \ \mathbf{dispose}\ y; \\ \ \mathrm{GET} \\ \ \mathbf{dispose}\ y) \end{array}$$

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

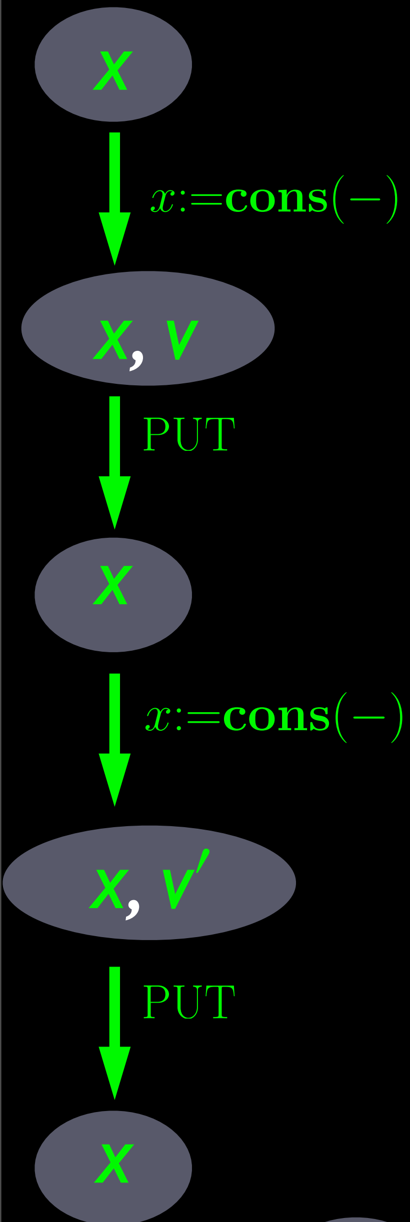
A LOCAL PARALLEL COMPUTATION



$$
\begin{array}{l} (x{:=}\mathbf{cons}(-); \\ \ \mathrm{PUT}; \\ \ x{:=}\mathbf{cons}(-); \\ \ \mathrm{PUT}) \end{array}
\;\Big\|\;
\begin{array}{l} (\mathrm{GET}; \\ \ \mathbf{dispose}\ y; \\ \ \mathrm{GET} \\ \ \mathbf{dispose}\ y) \end{array}
$$

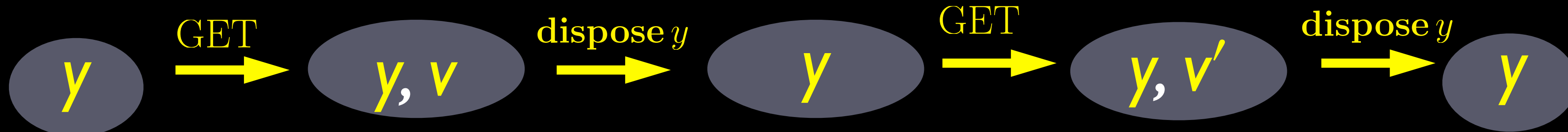
$$\Gamma \;=\; \mathit{buf}(c, \mathit{full}) : (\mathit{full} \wedge c \mapsto \_) \vee (\neg \mathit{full} \wedge \mathbf{emp}$$

A LOCAL PARALLEL COMPUTATION



$$\left(\begin{array}{l} x{:=}\mathbf{cons}(-);\\ \mathrm{PUT};\\ x{:=}\mathbf{cons}(-);\\ \mathrm{PUT}\end{array}\right) \;\Big\|\; \left(\begin{array}{l}\mathrm{GET};\\ \mathbf{dispose}\ y;\\ \mathrm{GET}\\ \mathbf{dispose}\ y\end{array}\right)$$

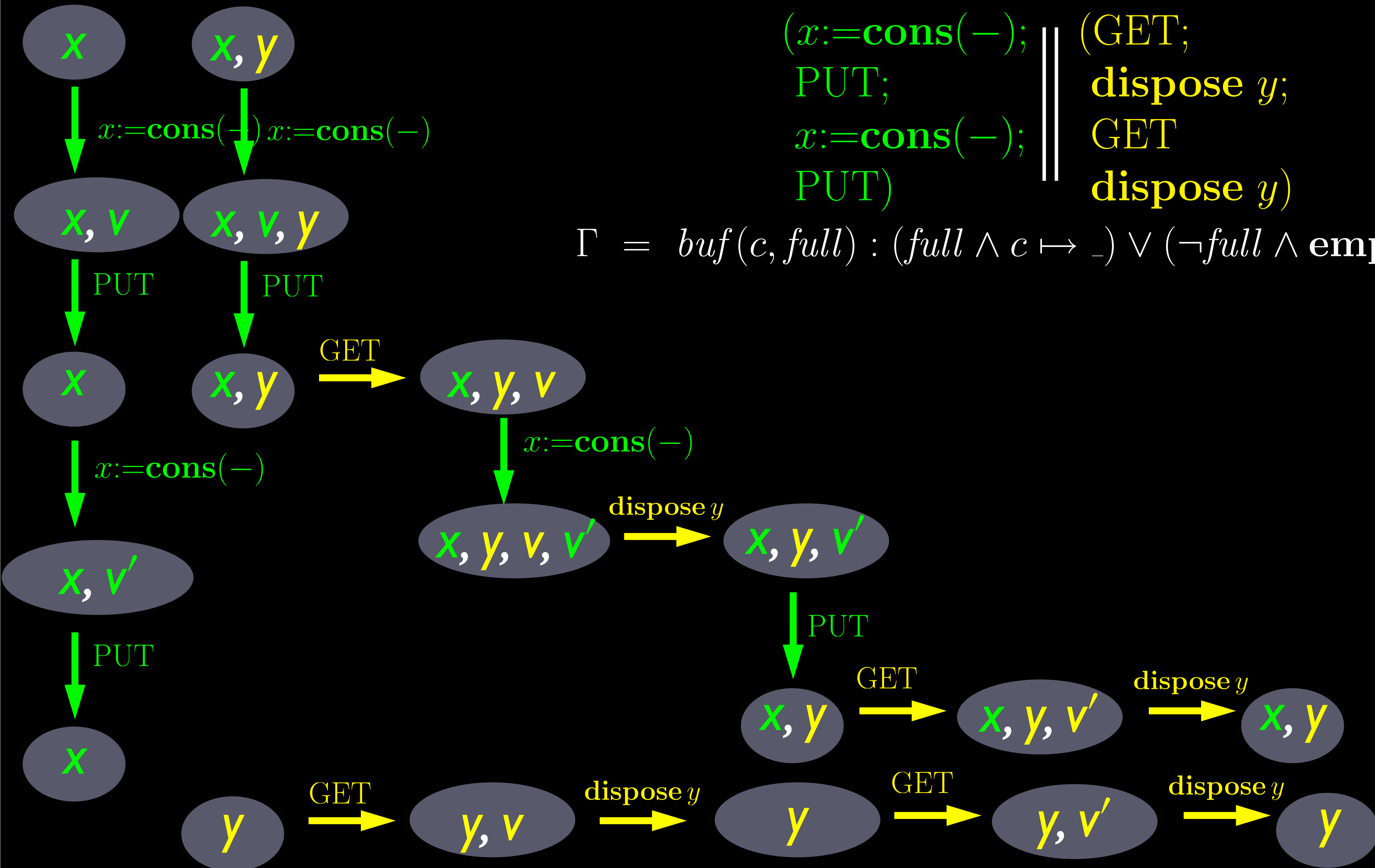
$$\Gamma \;=\; \mathit{buf}(c, \mathit{full}) : (\mathit{full} \wedge c \mapsto \_) \vee (\neg \mathit{full} \wedge \mathbf{emp}$$



A LOCAL PARALLEL COMPUTATION

$$(x{:=}\mathbf{cons}(-);\ \mathrm{PUT};\ x{:=}\mathbf{cons}(-);\ \mathrm{PUT}) \parallel (\mathrm{GET};\ \mathbf{dispose}\ y;\ \mathrm{GET}\ \mathbf{dispose}\ y)$$

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



PARALLEL DECOMPOSITION

Assume

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

$$h = h_1 \cdot h_2, \ A = A_1 \cdot A_2$$

$$\alpha \in \alpha_1 \ {}_{A_1}\|_{A_2} \, \alpha_2$$

If

$$\neg(s_1, h_1, A_1) \xrightarrow[\Gamma]{\alpha_1} \textbf{abort}$$

$$\neg(s_2, h_2, A_2) \xrightarrow[\Gamma]{\alpha_2} \textbf{abort}$$

then

$$\exists h_1', h_2', A_1', A_2'.$$

$$h' = h_1' \cdot h_2', \ A' = A_1' \cdot A_2',$$

$$(s_1, h_1, A_1) \xrightarrow[\Gamma]{\alpha_1} (s_1', h_1', A_1')$$

$$(s_2, h_2, A_2) \xrightarrow[\Gamma]{\alpha_2} (s_2', h_2', A_2')$$

CONNECTION

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

Show that 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

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

CONNECTION LEMMA

Suppose

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

If

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

$$(s \backslash owned(\Gamma), h_1) \xrightarrow[\Gamma]{\alpha} (s_1', h_1')$$

then

$$s_1' = s' \backslash owned(\Gamma)$$

$$\exists h_2'.\ h' = h_1' \cdot h_2' \ \&\ (s', h_2') \models inv(\Gamma)$$

DECOMPOSITION LEMMA

If

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

$$\neg\ (s_1, h_1, A_1) \xrightarrow[\Gamma]{\alpha_1} \textbf{abort}$$

$$\neg\ (s_2, h_2, A_2) \xrightarrow[\Gamma]{\alpha_2} \textbf{abort}$$

then

$$(s_1, h_1, A_1) \xrightarrow[\Gamma]{\alpha_1} (s_1', h_1', A_1')$$

$$(s_2, h_2, A_2) \xrightarrow[\Gamma]{\alpha_2} (s_2', h_2', A_2')$$

$$h' = h_1' \cdot h_2',\ \ A' = A_1' \cdot A_2'$$

COROLLARY

If

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

$$h = h_1 \cdot h_2$$

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

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

then

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

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

$$h' = h_1' \cdot h_2'$$

... hence Parallel Rule is sound

COROLLARY

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

Validity implies error-freedom:

Every finite *computation* of c
from a global state satisfying
$$p * inv(\Gamma)$$
is error-free,
and ends in a state satisfying
$$q * inv(\Gamma)$$

cf. traditional notion of validity

LOSING PRECISION

$$r : \mathbf{true} \vdash \{\mathbf{emp} \lor \mathbf{one}\}\mathbf{with}\ r\ \mathbf{do}\ \mathbf{skip}\{\mathbf{emp}\}$$

INVALID

but would be provable
if we drop precision constraint

Reynolds

BEYOND PRECISION

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

R supported

p,*q* intuitionistic

INTUITIONISTIC

p is intuitionistic if
for all states (s, h)
if $(s, h) \models p$ and $h \subseteq h'$

then $(s, h') \models p$

SUPPORTED

p is supported if
for all states (s, h)
if $(s, h) \models p$
there is a unique minimal $h' \subseteq h$

such that $(s, h') \models p$

SOUNDNESS

- Modify local semantics
  - transfer *minimal* heap
- Makes no change if *R* precise
- Soundness proof still works
- Can use precise or supported/intuitionistic

CONCLUSIONS

- Concurrent separation logic
  - generalizes Owicki-Gries, Hoare
- Traces + local semantics
  - models ownership transfer, loose coupling
  - yields soundness proof
  - embodies Dijkstra's Principle

FUTURE WORK

- Deadlock, total correctness, safety, liveness
- Monitors, more general semaphores
- Passification
  - semantics already treats store, heap alike
- Concurrent procedures
  - Parallel Algol + pointers?