CONCURRENT SEPARATION LOGIC... semantics and soundness

Stephen Brookes

Carnegie Mellon University

Tutorial

MFPS 2006

Outline

- Denotational semantics
  - parallel programs with shared mutable state
- Logic
  - resource-sensitive partial correctness
- Soundness
  - ownership transfer
  - every provable formula is race-free

Semantics

- A *process* denotes a set of *traces*
- A trace is a sequence of *actions*
- Actions have *effect* on *state*
- State = store + heap + resources

A trace represents a
fair interactive computation

Actions

- Idle δ
- Store $i{=}v$ $i{:=}v$
- Heap $[l]{=}v$ $[l]{:=}v$ $alloc(l, L)$ $disp(l)$
- Resource $try(r)$ $acq(r)$ $rel(r)$
- Error $abort$

i ranges over program variables, r over resource names, v over integers, l over heap cells, L over lists of integers

Traces

A trace is a finite or infinite sequence of actions

- Let Λ be the set of actions
- The set of traces is $\mathbf{Tr} = \Lambda^* \cup \Lambda^\omega$

λ *will range over* Λ

$\alpha, \beta, \gamma, \rho$ *will range over* $\mathbf{Tr}$

Semantic functions

$$[\![e]\!] \subseteq \mathbf{Tr} \times V_{\text{int}}$$

$$[\![E]\!] \subseteq \mathbf{Tr} \times V_{\text{int}}^{*}$$

$$[\![b]\!] \subseteq \mathbf{Tr} \times V_{\text{bool}}$$

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

defined denotationally...

Semantics of expressions

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

$$[\![i]\!] = \{(i{=}v,\ v) \mid v \in V_{int}\}$$

$$[\![e_1{+}e_2]\!] = \{(\rho_1\rho_2,\ v_1{+}v_2) \mid (\rho_1,\ v_1) \in [\![e_1]\!] \ \&\ (\rho_2,\ v_2) \in [\![e_2]\!]\}$$

Semantics of expressions

$$[\![b]\!] \subseteq \mathbf{Tr} \times V_{\mathrm{bool}}$$

$$[\![\mathbf{true}]\!] = \{(\delta, \mathit{true})\}$$

$$[\![\mathbf{not}\ b]\!] = \{(\rho, \neg t) \mid (\rho, t) \in [\![b]\!]\}$$

Semantics of commands

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

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

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

$$[\![c_1;c_2]\!] = \{\, \alpha_1\alpha_2 \mid \alpha_1 \in [\![c_1]\!],\ \alpha_2 \in [\![c_2]\!] \,\}$$

concatenation

Semantics of commands

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

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

iteration

where

$$[\![b]\!]_{\text{true}} =_{\text{def}} \{\rho \mid (\rho, \mathit{true}) \in [\![b]\!]\}$$

$$[\![b]\!]_{\text{false}} =_{\text{def}} \{\rho \mid (\rho, \mathit{false}) \in [\![b]\!]\}$$

Semantics of commands

HEAP COMMANDS

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

$$[\![\ [e_1]{:=}e_2\ ]\!] = \{\rho_1\rho_2\ [v_1]{:=}v_2 \mid (\rho_1,v_1) \in [\![e_1]\!],\ (\rho_2,v_2) \in [\![e_2]\!]\}$$

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

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

Semantics of commands

CONDITIONAL CRITICAL REGIONS

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

where

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

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

Semantics of commands

LOCAL RESOURCE BLOCKS

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

obtained by erasing actions on r

from traces in which r is interference-free

A local channel cannot be accessed outside the scope

Semantics of commands

PARALLEL COMPOSITION

$$[\![c_1 \parallel c_2]\!] = [\![c_1]\!] \,{}_{\varnothing}\!\parallel_{\varnothing}\, [\![c_2]\!]$$

resource-sensitive, race-detecting, fair interleaving

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

parallel composition of traces, with disjoint sets of resources

Parallel composition

$$\lambda_1\beta_1 \underset{A_1\,A_2}{||} \lambda_2\beta_2$$

mutual exclusion for resources

$$=_{\text{def}} \{ \lambda_1\beta \mid \beta \in \beta_1 \underset{A_1'\,A_2}{||} \lambda_2\beta_2 \;\&\; (A_1,A_2) \xrightarrow{\lambda_1} (A_1',A_2) \}$$

$$\cup \{ \lambda_2\beta \mid \beta \in \lambda_1\beta_1 \underset{A_1\,A_2'}{||} \beta_2 \;\&\; (A_2,A_1) \xrightarrow{\lambda_2} (A_2',A_1) \}$$

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

race causes error

Interference

concurrent write to same variable or heap cell

$\lambda_1 \bowtie \lambda_2$ iff $\mathrm{free}(\lambda_1) \cap \mathrm{writes}(\lambda_2) \neq \emptyset$ **or** $\mathrm{writes}(\lambda_1) \cap \mathrm{free}(\lambda_2) \neq \emptyset$

Mutual exclusion

process with A_1 can do λ in environment with A_2

$(A_1, A_2) \xrightarrow{acq(r)} (A_1 \cup \{r\}, A_2)$ iff $r \notin A_1 \cup A_2$

$(A_1, A_2) \xrightarrow{rel(r)} (A_1 \setminus r, A_2)$ iff $r \in A_1$

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

Examples

$[\![x := 1 \parallel x := 2]\!] = \{x{:=}1\ x{:=}2,\ x{:=}2\ x{:=}1\} \cup \{abort \mid v \in \mathrm{V}_{\mathrm{int}}\}$

$[\![\textbf{with } r \textbf{ do } x := 1]\!] = try(r)^{\infty}\ acq(r)\ \{x{:=}1\}\ rel(r)$

$[\![\textbf{with } r \textbf{ do } x := 2]\!] = try(r)^{\infty}\ acq(r)\ \{x{:=}2\}\ rel(r)$

$[\![\textbf{resource } r \textbf{ in}$
$\textbf{with } r \textbf{ do } x := 1 \parallel \textbf{with } r \textbf{ do } x := 2]\!]$
$= \{x{:=}1\ x{:=}2,\ x{:=}2\ x{:=}1\}$

Actions and state

- So far we have kept state *implicit*
  - state = store + heap + resources
  - $\sigma = (s, h, A)$
- But actions have *effect*
  - when *enabled*
  - cause state change
  - may produce runtime error

Effects

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

$$(s, h, A) \overset{\lambda}{\Longrightarrow} \textbf{abort}$$

a *transition relation*

specifying

when the action λ is *enabled*
and its *effect* when executed

Effects

READ

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

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

WRITE

$$(s, h, A) \overset{i:=v}{\Longrightarrow} ([s|i:v], h, A) \quad \text{if } i \in dom(s)$$

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

Effects

LOOKUP

$$(s, h, A) \overset{[l]=v}{\Longrightarrow} (s, h, A) \quad \text{if } (l,v) \in h$$

$$(s, h, A) \overset{[l]=v}{\Longrightarrow} \textbf{abort} \quad \text{if } l \notin \mathit{dom}(h)$$

UPDATE

$$(s, h, A) \overset{[l]:=v}{\Longrightarrow} (s, [h|l:v], A) \quad \text{if } l \in \mathit{dom}(h)$$

$$(s, h, A) \overset{[l]:=v}{\Longrightarrow} \textbf{abort} \quad \text{if } l \notin \mathit{dom}(h)$$

Effects

ALLOCATE

$$(s, h, A) \overset{alloc(l,[v_0,\ldots,v_n])}{\Rightarrow} (s, h', A)$$

if $\{l, l+1, \ldots, l+n\} \cap \mathrm{dom}(h) = \emptyset$
and $h' = [h \mid l{:}v_0, \ldots, l+n{:}v_n]$

DISPOSE

$$(s, h, A) \overset{disp(l)}{\Rightarrow} (s, h \backslash l, A) \quad \text{if } l \in \mathrm{dom}(h)$$

$$(s, h, A) \overset{disp(l)}{\Rightarrow} \textbf{abort} \quad \text{if } l \notin \mathrm{dom}(h)$$

Effects

RESOURCE ACTIONS

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

$$(s, h, A) \overset{rel(r)}{\Rightarrow} (s, h, A \backslash r) \quad \text{if } r \in A$$

$$(s, h, A) \overset{try(r)}{\Rightarrow} (s, h, A) \quad \text{always}$$

Effects

IDLE

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

ERROR

$$(s, h, A) \overset{abort}{\Rightarrow} \textbf{abort}$$

$$\textbf{abort} \overset{\lambda}{\Rightarrow} \textbf{abort}$$

Respect for resources

THEOREM

If $\alpha \in [\![c]\!]$

and

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

then $dom(s) = dom(s')$

and $A' = \varnothing$

We usually omit $\varnothing$

and write $(s,h) \overset{\alpha}{\Rightarrow} (s',h')$

Race-free programs

DEFINITION

c **is *race-free* from** (s, h)

iff

$$\neg\, \exists \alpha \in [\![c]\!].(s, h) \overset{\alpha}{\Rightarrow} \textbf{abort}$$

every execution of c from (s, h) is error-free

Examples

PUT :: **with** r **when** full=0 **do** (z:=x; full:=1)
GET :: **with** r **when** full=1 **do** (y:=z; full:=0)

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

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

$\mathrm{PUT}^M \,||\, \mathrm{GET}^N \quad (M,N > 0)$

are race-free from (s,h) if

$\{x,y,z\} \subseteq \mathrm{dom}(s)$ & $s(x) \in \mathrm{dom}(h)$ & $s(full)=0$

$(s,h) \models full=0 \wedge x \mapsto \_$

Summary

- Trace semantics, defined denotationally, supports compositional reasoning
  - Treats a potential race as catastrophe
- Suitable for race-free partial correctness, total correctness, safety and liveness

Will provide foundation for concurrent separation logic…

Concurrent separation logic

- A more formal treatment of the logic underlying Peter O'Hearn's work
- Syntax and semantics
- Proof of soundness

Key notions:

rely/guarantee
ownership transfer

Syntax

Resource-sensitive partial correctness formulas

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

where

Γ is a *resource context*

$$r_1(X_1):R_1, \ldots r_n(X_n):R_n$$

protection list

resource invariant

subject to well-formedness constraints…

Notation

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

- $\mathrm{dom}(\Gamma) = \{r_1, \ldots, r_k\}$ — *resource names*
- $\mathrm{owned}(\Gamma) = X_1 \cup \ldots \cup X_k$ — *protected variables*
- $\mathrm{free}(\Gamma) = \mathrm{free}(R_1) \cup \ldots \cup \mathrm{free}(R_k)$ — *variables mentioned in resource invariants*
- $\mathrm{inv}(\Gamma) = R_1 \star \ldots \star R_k$ — *separate conjunction of resource invariants*

Well-formed formulas

$r_1(X_1):R_1, \ldots r_k(X_k):R_k \vdash \{p\}c\{q\}$

is *well-formed* iff

pre- and post-conditions don't mention protected variables

- R_i is *precise* & free(p,q)$\cap X_i = \emptyset$ for all i
- $r_i \neq r_j$, $X_i \cap X_j = \emptyset$, free(R_i)$\cap X_j = \emptyset$ for i ≠ j

protected variables are partitioned among the resources

Precision

DEFINITION

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

FACTS

emp, $e \mapsto e'$ *are precise*

if p_1 *and* p_2 *are precise, so are*

$p_1 \ast p_2$

$(b \wedge p_1) \vee (\neg b \wedge p_2)$

Inference rules

- Syntax-directed rules
  - for each program construct
- Structural rules
  - rule of consequence, auxiliary variables
- Static side conditions
  - impose well-formedness constraints

Parallel

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

provided

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

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

$\mathrm{free}(c_1) \cap \mathrm{writes}(c_2) \subseteq \mathrm{owned}(\Gamma)$

$\mathrm{free}(c_2) \cap \mathrm{writes}(c_1) \subseteq \mathrm{owned}(\Gamma)$

Region

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

provided

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

R precise

Resource

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

Frame

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

provided

$\text{free}(I) \cap \text{writes}(c) = \emptyset$

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

Validity

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

iff

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

... needs to be formalized!

Separation Principle

At all stages of execution:

- For each available resource, the invariant holds, separately
- The heap is partitioned among processes and available resources
  - ownership transfers when resource is acquired or released

Formalized using
local states, local transitions...

Local states

A state is *local for* Γ if it provides sufficient resources for its protected variables...

(s, h, A)
is local for Γ
iff
$\mathrm{dom}(s) \cap \mathrm{owned}(\Gamma) = \mathrm{owned}(\Gamma \upharpoonright A)$

the piece of global state visible to a process with resources A that respects Γ

Local transitions

- Interactive computation
  in an environment that respects Γ
  - ownership transfer
  - protection rules
  - preservation of invariants

the local view
of a global computation
as seen by a process

global and **local views**

A global state (s,h,A)
in which processes own
(h_1,A_1), ..., (h_n,A_n)
and resource invariants
hold separately, in h′

$A = A_1 \cup \ldots \cup A_n$
$h = h_1 \cup \ldots \cup h_n \cup h'$
$(s,h') \models inv(\Gamma \backslash A)$

local states
$(s \backslash owned(\Gamma \backslash A_1), h_1, A_1)$,
...
$(s \backslash owned(\Gamma \backslash A_n), h_n, A_n)$

Local transitions

READ

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

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

WRITE

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

if $i \in dom(s) - free(\Gamma \setminus A)$

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

if $i \notin dom(s)$ or $i \in free(\Gamma \setminus A)$

Local transitions

LOOKUP

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

$$(s, h, A) \xrightarrow{[l]=v}_{\top} \textbf{abort} \quad \text{if } l \notin \mathit{dom}(h)$$

UPDATE

$$(s, h, A) \xrightarrow{[l]:=v} (s, [h|l:v], A) \quad \text{if } l \in \mathit{dom}(h)$$

$$(s, h, A) \xrightarrow{[l]:=v}_{\top} \textbf{abort} \quad \text{if } l \notin \mathit{dom}(h)$$

Local transitions

ALLOCATE

$$(s, h, A) \xrightarrow{alloc(l,[v_0,\ldots,v_n])} (s, h', A)$$

if $\{l, l+1, \ldots, l+n\} \cap \mathrm{dom}(h) = \emptyset$

and $h' = [h \mid l{:}v_0, \ldots, l+n{:}v_n]$

DISPOSE

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

$$(s, h, A) \xrightarrow{disp(l)} \textbf{abort} \quad \text{if } l \notin \mathrm{dom}(h)$$

Local transitions

non-deterministic!

ACQUIRE

$$(s, h, A) \xrightarrow{acq(r)}_{\Gamma, r(X):R} (s \cup s', h \cup h', A \cup \{r\})$$

$$\text{if } r \notin A,\ \mathbf{dom}(s') = X,\ h \perp h',\ (s \cup s', h') \models R$$

process assumes that
environment preserves invariants

and claims ownership of
variables protected by r
+ heap satisfying the invariant

Local transitions

RELEASE

R precise... so at most one *h′*

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

$$(s, h, A) \xrightarrow[\Gamma, r(X):R]{rel(r)} \textbf{abort}$$
$$\text{if } r \in A, \forall h' \subseteq h.\ (s, h') \models \neg R$$

process guarantees to preserve invariants

and cedes ownership of
variables protected by r
+ heap satisfying invariant

Local transitions

IDLE

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

ERROR

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

$$\textbf{abort} \xrightarrow[\Gamma]{\lambda} \textbf{abort}$$

Local computation

$$\sigma \xrightarrow[\Gamma]{\lambda_1 \ldots \lambda_n} \sigma'$$

means

$$\sigma \xrightarrow[\Gamma]{\lambda_1} \cdots \xrightarrow[\Gamma]{\lambda_n} \sigma'$$

$$\sigma \xrightarrow[\Gamma]{\lambda_1 \ldots \lambda_n} \text{abort}$$

means

$$\sigma \xrightarrow[\Gamma]{\lambda_1} \cdots \xrightarrow[\Gamma]{\lambda_n} \text{abort}$$

Example

A local computation of PUT using Γ

Γ:: buf(z,full): (full=0 ∧ emp) ∨ (full=1 ∧ z↦-)

$$([x:v_0],[v_0:v],\varnothing) \xrightarrow[\Gamma]{acq(buf)} ([x:v_0,full:0,z:v_1],[v_0:v],\{buf\})$$

$$\xrightarrow[\Gamma]{\beta} ([x:v_0,full:1,z:v_0],[v_0:v],\{buf\})$$

$$\xrightarrow[\Gamma]{rel(buf)} ([x:v_0],[\,],\varnothing)$$

where

β : full=0 x=v_0 z:=v_0 full:=1

Example

A local computation of GET using Γ

Γ:: buf(z,full): (full=0 ∧ emp) ∨ (full=1 ∧ $z^{\mapsto}$-)

$$([y:v_2],[\ ],\varnothing) \xrightarrow[\Gamma]{acq(buf)} ([y:v_2,full:1,z:v_0],[v_0:v],\{buf\})$$

$$\xrightarrow[\Gamma]{\gamma} ([y:v_0,full:0,z:v_0],[v_0:v],\{buf\})$$

$$\xrightarrow[\Gamma]{rel(buf)} ([y:v_0],[v_0:v],\varnothing)$$

where

γ : full=1 z=v_0 y:=v_0 full:=0

Example

A local computation of PUT using Γ′

Γ′:: buf(z,full): (full=0 ∧ emp) ∨ (full=1 ∧ emp)

$$([x{:}v_0],[v_0{:}v],\varnothing) \xrightarrow[\Gamma']{\text{acq(buf)}} ([x{:}v_0,\text{full}{:}0,z{:}v_1],[v_0{:}v],\{buf\})$$

$$\xrightarrow[\Gamma']{\beta} ([x{:}v_0,\text{full}{:}1,z{:}v_0],[v_0{:}v],\{buf\})$$

$$\xrightarrow[\Gamma']{\text{rel(buf)}} ([x{:}v_0],[v_0{:}v],\varnothing)$$

where

β : full=0 x=v_0 z:=v_0 full:=1

GET

A local computation of GET using Γ′

Γ′:: buf(z,full): (full=0 ∧ emp) ∨ (full=1 ∧ emp)

$([y:v_2],[\ ],\emptyset) \xrightarrow[\Gamma']{acq(buf)} ([y:v_2,full:1,z:v_0],[\ ],\{buf\})$

$\xrightarrow[\Gamma']{\gamma} ([y:v_0,full:0,z:v_0],[\ ],\{buf\})$

$\xrightarrow[\Gamma']{rel(buf)} ([y:v_0],[\ ],\emptyset)$

where

γ : full=1 z=v_0 y:=v_0 full:=0

Local computations

- Execution in an environment that respects protection rules and maintains invariants
- Abort if process breaks protection rules or fails to guarantee invariant

A formalization of dynamic ownership transfer

Validity

DEFINITION

$\Gamma \vdash \{p\}c\{q\}$ is *valid*

iff

$\forall \alpha \in [\![c]\!].\forall \sigma$: dom($\sigma$)$\supseteq$free(c,$\Gamma$)-owned($\Gamma$).

$\sigma \models p \;\&\; \sigma \overset{\alpha}{\underset{\Gamma}{\Rightarrow}} \sigma'$ implies $\sigma' \models q$

$\sigma' \neq$ **abort**

A formal definition
matching the intuition given earlier

Examples

Γ:: buf(z,full): (full=0 ∧ emp) ∨ (full=1 ∧ $z^{\mapsto}$-)

Γ⊢{$x^{\mapsto}$-}PUT{emp}
Γ⊢{emp}GET{$y^{\mapsto}$-}
valid

Γ′:: buf(z,full): (full=0 ∧ emp) ∨ (full=1 ∧ emp)

Γ′⊢{$x^{\mapsto}$-}PUT{$x^{\mapsto}$-}
Γ′⊢{emp}GET{emp}
valid

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

Γ:: buf(z,full): (full=0 ∧ emp) ∨ (full=1 ∧ $z \mapsto -$)

$x \mapsto -$

$$([x:v_0],[v_0:v],\varnothing) \xrightarrow[\Gamma]{\mathrm{acq(buf)}} ([x:v_0,\mathrm{full}:0,z:v_1],[v_0:v],\{\mathrm{buf}\})$$

$$\xrightarrow[\Gamma]{\beta} ([x:v_0,\mathrm{full}:1,z:v_0],[v_0:v],\{\mathrm{buf}\})$$

$$\xrightarrow[\Gamma]{\mathrm{rel(buf)}} ([x:v_0],[\,],\varnothing)$$

emp

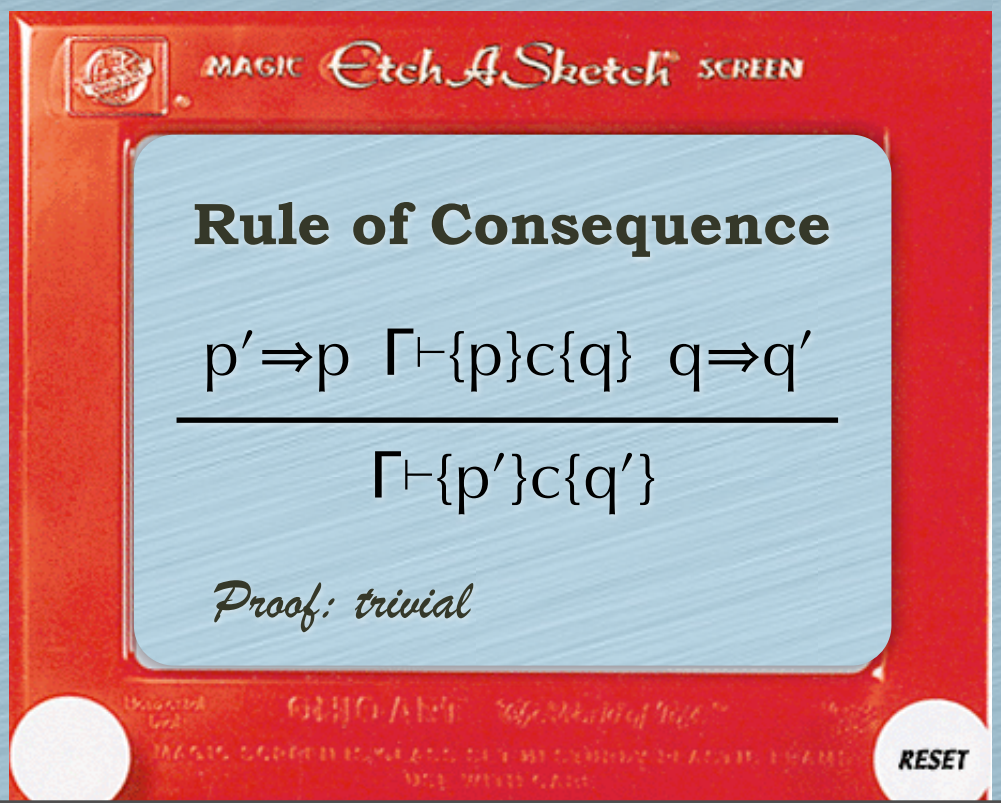
Soundness Theorem

- Concurrent separation logic is sound
  - every provable formula is valid

Proof:
Each inference rule preserves validity
when its side conditions hold

Proof sketch

- We consider the rules for Parallel, Region, Local Resource, Frame
- Other rules are similar



Parallel rule

$\Gamma \vdash \{p_1\}c_1\{q_1\}$ valid

$\Gamma \vdash \{p_2\}c_2\{q_2\}$ valid

1

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

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

2

$\mathrm{free}(c_1) \cap \mathrm{writes}(c_2) \subseteq \mathrm{owned}(\Gamma)$

$\mathrm{free}(c_2) \cap \mathrm{writes}(c_1) \subseteq \mathrm{owned}(\Gamma)$

3

PARALLEL DECOMPOSITION Lemma

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

Parallel Decomposition Lemma

Suppose

$\text{free}(\alpha_1) \cap \text{writes}(\alpha_2) \subseteq \text{owned}(\Gamma)$
$\text{free}(\alpha_2) \cap \text{writes}(\alpha_1) \subseteq \text{owned}(\Gamma)$
$\alpha \in \alpha_1 || \alpha_2, \quad h = h_1 \cup h_2, \; h_1 \perp h_2$

❸

If

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

then

$(s \backslash \text{writes}(\alpha_2), h_1) \xrightarrow[\Gamma]{\alpha_1} \text{abort}$

or

$(s \backslash \text{writes}(\alpha_1), h_2) \xrightarrow[\Gamma]{\alpha_2} \text{abort}$

If

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

$(s \backslash \text{writes}(\alpha_2), h_1) \xrightarrow[\Gamma]{\alpha_1} (s_1',h_1')$

$(s \backslash \text{writes}(\alpha_1), h_2) \xrightarrow[\Gamma]{\alpha_2} (s_2',h_2')$

then

$h' = h_1' \cup h_2', \; h_1' \perp h_2'$
$s_1' = s' \backslash \text{writes}(\alpha_2)$
$s_2' = s' \backslash \text{writes}(\alpha_1)$

Soundness of Par

- Suppose $(s,h) \models p_1 \ast p_2$, $(s,h) \xrightarrow[\Gamma]{\alpha} \sigma'$
- Choose $h_1 \perp h_2$ so that $h = h_1 \cup h_2$, $(s,h_1) \models p_1$, $(s,h_2) \models p_2$
- Then $(s\backslash writes(\alpha_2), h_1) \models p_1$, $(s\backslash writes(\alpha_1), h_2) \models p_2$ by ❷
- By Decomposition Lemma, ❶, ❸,

  $\sigma' = (s', h')$ $h' = h_1' \cup h_2'$ $h_1' \perp h_2'$

  $(s\backslash writes(\alpha_2), h_1) \xrightarrow[\Gamma]{\alpha_1} (s'\backslash writes(\alpha_2), h_1') \models q_1$

  $(s\backslash writes(\alpha_1), h_2) \xrightarrow[\Gamma]{\alpha_2} (s'\backslash writes(\alpha_1), h_2') \models q_2$
- Hence $(s',h_1') \models q_1$ and $(s',h_2') \models q_2$ by ❷
- So $\sigma' \models q_1 \ast q_2$ as required

Region rule

$\Gamma \vdash \{(p*R)\wedge b\}c\{q*R\}$ valid

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

R precise

REGION LEMMA

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

Soundness of region rule

⟦**with** r **when** b **do** c⟧ = acq(r) ⟦b⟧$_{true}$ ⟦c⟧ rel(r) …

- Let $(s,h) \models p$, $\beta \in [\![b]\!]_{true}$, $\gamma \in [\![c]\!]$
- If $(s,h) \xrightarrow[\Gamma,r(X):R]{acq(r)\ \beta\gamma\ rel(r)} \sigma'$

  there are (s_0,h_0), (s_0',h_0'), $h' \perp h_0'$ such that

$$(s,h) \xrightarrow[\Gamma,r(X):R]{acq(r)} (s \cup s_0,\ h \cup h_0) \models (p * R) \wedge b$$

$$\xrightarrow[\Gamma,r(X):R]{\beta\gamma} (s' \cup s_0',\ h' \cup h_0') \models q * R$$

$$\xrightarrow[\Gamma,r(X):R]{rel(r)} (s',\ h') = \sigma' \models q$$

as required

precision is crucial!

Resource rule

$\Gamma, r(X):R \vdash \{p\}c\{q\}$
valid

RESOURCE LEMMA

$\Gamma \vdash \{p*R\}$**resource** r **in** $c\{q*R\}$
valid

Soundness of resource rule

$[\![\text{resource r in c}]\!] = \{\beta \backslash r \mid \beta \in [\![c]\!]_r\}$

- Let $(s,h) \models p \star R$, $\beta \in [\![c]\!]_r$, $(s,h) \xrightarrow[\Gamma]{\beta \backslash r} \sigma'$
- Choose $h_1 \perp h_2$ such that

  $(s,h_1) \models p, \quad (s,h_2) \models R, \quad h = h_1 \cup h_2$
- Then there must be $h_1' \perp h_2'$ such that

  $(s,h_1) \xrightarrow[\Gamma,r(X):R]{\beta} (s',h_1') \quad \sigma' = (s', h_1' \cup h_2')$

  $(s',h_1') \models q \quad (s',h_2') \models R$
- Hence $\sigma' \models q \star R$ as required

Frame rule

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

$\text{free}(I) \cap \text{writes}(c) = \emptyset$

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

FRAME LEMMA

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

Frame Lemma

Suppose
$h = h_1 \cup h_2,\ h_1 \perp h_2$

If

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

then

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

If

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

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

then

$s_1' = s' \quad h_1' \perp h_2$

$h' = h_1' \cup h_2$

Soundness of Frame rule

- Let $\alpha \in [\![c]\!]$, $(s,h) \models p \ast I$, $(s,h) \xrightarrow[\Gamma]{\alpha} \sigma'$
- Choose $h_1 \perp h_2$ so that $h = h_1 \cup h_2$ and
  $(s,h_1) \models p$, $(s,h_2) \models I$
- Since $\Gamma \vdash \{p\}c\{q\}$ is valid, there are s', h_1' such that $(s,h_1) \xrightarrow[\Gamma]{\alpha} (s',h_1')$ $\models q$
- By Frame Lemma $h_1' \perp h_2$ $\sigma' = (s', h_1' \cup h_2')$
- Since $\mathrm{free}(p) \cap \mathrm{writes}(\alpha) = \varnothing$, $(s',h_2) \models I$
- Hence $\sigma' \models q \ast I$ as required

And so on...

- For each inference rule,
  a similar proof case

Summary:
every provable formula is valid

Validity, revisited

- We defined validity in terms of *local* states and *local* computation
- Need to connect with *global* notions
  - independent of the logic
  - related to standard operational semantics
  - show that provable programs are guaranteed to be race-free

Global/local lemma

Suppose $h = h_1 \cup h_2$, $h_1 \perp h_2$, $(s,h_2) \models inv(\Gamma)$

If $(s,h) \overset{\alpha}{\Rightarrow}$ **abort**

then $(s \backslash owned(\Gamma), h_1) \overset{\alpha}{\underset{\Gamma}{\rightarrow}}$ **abort**

If $(s,h) \overset{\alpha}{\Rightarrow} (s',h')$

and $\neg\ (s \backslash owned(\Gamma), h_1) \overset{\alpha}{\underset{\Gamma}{\rightarrow}}$ **abort**

then there are $h_1' \perp h_2'$ such that

$(s \backslash owned(\Gamma), h_1) \overset{\alpha}{\underset{\Gamma}{\rightarrow}} (s' \backslash owned(\Gamma), h_1')$

$h' = h_1' \cup h_2'$, $(s',h_2') \models inv(\Gamma)$

Corollary

If $\Gamma \vdash \{p\}c\{q\}$ is valid,

$\forall \alpha \in [\![c]\!].$

$\forall \sigma: \mathrm{dom}(\sigma) \supseteq \mathrm{free}(c).\ \forall \sigma'.$

$\sigma \vDash p \ast \mathrm{inv}(\Gamma)\ \&\ \sigma \overset{\alpha}{\Rightarrow} \sigma'$

implies

$\sigma' \vDash q \ast \mathrm{inv}(\Gamma)$

provable implies race-free


When you gather all
your resources together,
goals are accomplished

Precision is crucial

Region rule is unsound
with imprecise invariants

valid

$$\frac{\vdash\{(\mathrm{emp} \vee x\mapsto\_) \ast \mathrm{true}\}\ \textbf{skip}\ \{\mathrm{emp} \ast \mathrm{true}\}}{r{:}\mathrm{true} \vdash \{\mathrm{emp} \vee x\mapsto\_\}\ \textbf{with}\ r\ \textbf{do skip}\ \{\mathrm{emp}\}}$$

invalid

(Example due to John Reynolds)