MFPS, May 2006

A semantics for concurrent permission logic

Stephen Brookes
CMU

Traditional logic

Owicki/Gries '76

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

- Shared-memory parallel programs
- Resource-sensitive partial correctness

Γ of form $r_1(X_1){:}R_1, \ldots r_n(X_n){:}R_n$

$$inv(\Gamma) =_{def} R_1 \wedge \ldots \wedge R_n$$

$$owned(\Gamma) =_{def} X_1 \cup \ldots \cup X_n$$

resource names, protection lists, invariants

(subject to static constraints)

Inference rules

Owicki/Gries

if critical variables are protected

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

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

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

static constraints ensure race-freedom

Validity

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

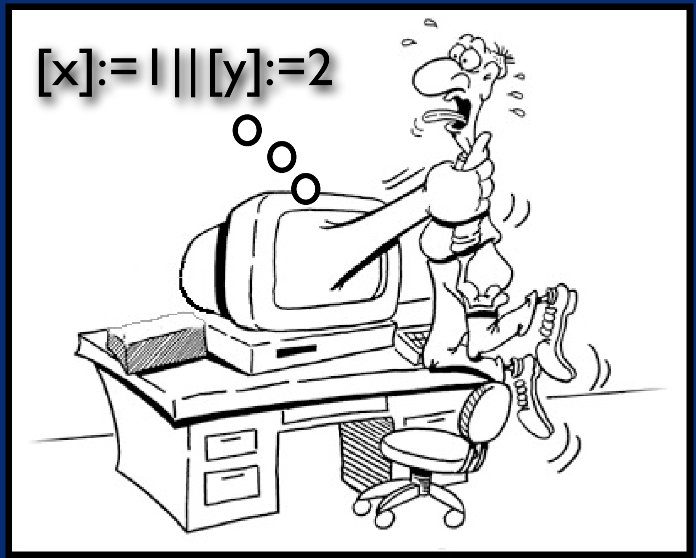
Every finite computation of c
in an environment that respects Γ,
from a state satisfying $p \wedge \mathit{inv}(\Gamma)$,

respects Γ, is race-free,
and ends in a state satisfying $q \wedge \mathit{inv}(\Gamma)$

(state = store)

Soundness

- Owicki-Gries logic is *sound*, for *simple* shared-memory programs
- But *unsound for pointer programs*
  - Static constraints don't prevent *heap races*



aliasing can't be detected statically

Concurrent separation logic

O'Hearn '04
Reynolds '02

- Combine Owicki-Gries with *separation logic*
- Use $\star$ to enforce mutual exclusion for *heap*
- Use *precise* resource invariants

$(s,h) \models \varphi_1 \star \varphi_2$
iff $\exists h_1 \perp h_2.\ h = h_1 \cup h_2$ &
$(s,h_1) \models \varphi_1$ & $(s,h_2) \models \varphi_2$

$inv(\Gamma) =_{def} R_1 \star \ldots \star R_n$

Each invariant holds separately, in a unique subheap

Inference rules

O'Hearn

if critical variables are protected

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

$\star$ for $\wedge$

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

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

Static constraints ensure race-freedom for variables...

... using $\star$ *prevents heap races*

Validity

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

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

$\star$ for $\wedge$

(state = store + heap)

Soundness

Brookes '04

THEOREM

Every provable formula is valid

PROOF

- Based on *action trace* semantics
- Resource invariants hold *separately*, for available resources
- *Ownership* of *heap* + *protected variables* is deemed to *transfer* when process acquires or releases resource

precision is crucial

Problems

- Concurrent separation logic is too rigid
- Cannot handle *concurrent reads* of heap cells

$\vdash \{z \mapsto 0\}\ x{:=}[z]\ ||\ y{:=}[z]\ \{z \mapsto 0 \wedge x{=}y{=}0\}$

valid but not provable

$\vdash \{z = 0\}\ x{:=}z\ ||\ y{:=}z\ \{z = 0 \wedge x{=}y{=}0\}$

valid, provable

Concurrent permission logic

Parkinson, Bornat, Calcagno '06

- Blend Owicki-Gries with *permission logic*
- Treat store and heap identically
  - augment state with *permissions*
- Use a *permissive* form of ⋆
  - allow concurrent *reads* but not *writes*

And eliminate "awkward" side conditions...

Summary of talk

- Concurrent permission logic is *sound*
- Can still use *action trace* semantics
- Soundness proof generalizes earlier proof
  - permissive analogue of *precision* plays key role

we focus on store,
but heap can be handled
in the same manner

Actions

Brookes '04

- δ — idle
- $x=v$ — read variable
- $x:=v$ — write variable
- *try*(*r*), *acq*(*r*), *rel*(*r*) — resource operations
- *abort* — error

λ ranges over actions

Semantics

Brookes '04

- A command c denotes a set $[\![c]\!]$ of *action traces*
- Defined by structural induction

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

concatenation

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

resource-sensitive, race-detecting, fair interleaving

α *ranges over traces*

Actions need permission

- Reading requires *any* permission
  - not necessarily exclusive
- Writing requires *total* permission
  - mutually exclusive

...such constraints
will be used to
ensure race-freedom...

Permissions

PBC '06

$(\mathcal{P}, \otimes, \top)$

- partial commutative cancellative semi-group
- $\top \otimes p$ undefined
- $p \otimes p' \neq p$

$p \# p'$ iff $p \otimes p'$ defined
compatibility

$\top$ allows *read/write*

$p \neq \top$ allows *read* only

Fractional permissions

Boyland

- $\mathcal{P} = (0,1] \cap \mathbb{Q}$
- $p \otimes p' = p + p'$ if in (0,1]
- $\top = 1$

1 is total,
any other fraction
allows read only

... satisfies the required properties

Stores

$$s : S = \mathrm{Ide} \rightharpoonup_{fin} V \times \mathcal{P}$$

- Store maps *program variables* to *(value, permission)* *pairs*
- Stores are *consistent* if they give *same value* and *compatible permissions*, for common variables

  $s \# s'$
- Consistent stores can be *combined*

  $s \star s'$

Permissive formulas

PBC '06

φ	::=	**emp**	empty
	|	$\mathrm{Own}_p(x)$	singleton
	|	$\varphi_1 \star \varphi_2$	separating conjunction
	|	$E_1 = E_2$	equality
	|	$\exists X.\varphi$	existential

+ *standard boolean connectives*

E : *value expressions*
p : *permission expressions*
X : *logical variables*
x : *program variables*

expressions are pure, may contain logical variables

States

state = store + interpretation

(for logical variables)

- $\sigma = (s, i)$
- $(s, i) \# (s', i')$ iff $s \# s'$ & $i = i'$ — *compatibility*
- $(s, i) \star (s', i) = (s \star s', i)$ — *composition*

logical variables
denote values or permissions

Satisfaction

PBC '06

$(s,i) \vDash \mathbf{emp}$ iff $s=\{\ \}$

$(s,i) \vDash \mathrm{Own}_p(x)$ iff $\exists v.\ s=\{(x, (v, |p|i))\}$

$\sigma \vDash \varphi$

defined inductively

$\sigma \vDash \varphi_1 \star \varphi_2$ iff

$\exists \sigma_1,\sigma_2.\ \sigma = \sigma_1 \star \sigma_2\ \&\ \sigma_1 \vDash \varphi_1\ \&\ \sigma_2 \vDash \varphi_2$

$\sigma \vDash E_1 = E_2$ iff

$|E_1|\sigma = |E_2|\sigma\ \&\ \mathit{free}(E_1,E_2) \subseteq \mathit{dom}(\sigma)$

Examples

$(p, q \in \mathcal{P})$

$\text{Own}_p(x) \star \text{Own}_q(x)$

true in (s,i)

iff

$p \# q$ & $\exists v.\ s(x) = (v, p \otimes q)$

$\text{Own}_{p \otimes q}(x)$

$x=3$

true in (s,i)

iff

$\exists p \in \mathcal{P}.\ s(x) = (3, p)$

Precision

DEFINITION

ϑ is *precise* iff for all σ there is at most one pair (σ_1, σ_2) such that $\sigma = \sigma_1 \star \sigma_2$ and $\sigma_1 \vDash \vartheta$

EXAMPLES

emp, $\mathrm{Own}_p(x)$ are precise

if ϑ_1, ϑ_2 are precise, so are

$\vartheta_1 \star \vartheta_2$, $(B \wedge \vartheta_1) \vee (\neg B \wedge \vartheta_2)$

Program formulas

PBC '06

$\Gamma \vdash_{vr} \{\varphi\}\ c\ \{\psi\}$

no protection lists

- Γ of form $r_1{:}\vartheta_1, \ldots, r_n{:}\vartheta_n$
- $\vartheta_1, \ldots, \vartheta_n$ precise
- $r_1, \ldots, r_n$ distinct
- φ, ψ arbitrary formulas

no static constraints

$inv(\Gamma) =_{def} \vartheta_1 \star \ldots \star \vartheta_n$

Inference rules

PBC '06

no static constraints

$$\frac{\Gamma \vdash_{vr} \{\varphi_1\}\ c_1\ \{\psi_1\} \qquad \Gamma \vdash_{vr} \{\varphi_2\}\ c_2\ \{\psi_2\}}{\Gamma \vdash_{vr} \{\varphi_1 \star \varphi_2\}\ c_1 \| c_2\ \{\psi_1 \star \psi_2\}}$$

$$\frac{\Gamma \vdash_{vr} \{(\varphi \star \theta) \wedge b\}\ c\ \{\psi \star \theta\} \qquad \varphi \star \theta \Rightarrow b{=}b}{\Gamma, r{:}\theta \vdash_{vr} \{\varphi\}\ \textbf{with}\ r\ \textbf{when}\ b\ \textbf{do}\ c\ \{\psi\}}$$

extra premiss implies permission for b

$$\frac{\Gamma, r{:}\theta \vdash_{vr} \{\varphi\}\ c\ \{\psi\}}{\Gamma \vdash_{vr} \{\varphi \star \theta\}\ \textbf{resource}\ r\ \textbf{in}\ c\ \{\psi \star \theta\}}$$

Assignment rule

PBC '06

not the usual substitution rule!

$$\overline{\Gamma \vdash_{vr} \{\mathrm{Own}\ \ (x) \star O \wedge X{=}e\}\ x{:=}e\ \{\mathrm{Own}\ \ (x) \star O \wedge x{=}X\}}$$

where

O ranges over *ownership claims*

$$\mathrm{Own}_{p_1}(x_1) \star \ldots \star \mathrm{Own}_{p_k}(x_k)$$

permission constraints are implied for free(e), x

Examples

CONCURRENT READS

$$\vdash_{vr} \{Own_{\top}(x) \star Own_{\top}(y) \star Own_q(z)\}$$
$$x{:=}z \;||\; y{:=}z$$
$$\{Own_{\top}(x) \star Own_{\top}(y) \star Own_q(z) \wedge x{=}y{=}z\}$$

valid, provable

CONCURRENT WRITES

$$\vdash_{vr} \{Own_{\top}(x) \star Own_{\top}(x)\}$$
$$x{:=}x{+}1 \;||\; x{:=}x{+}1$$
$$\{Own_{\top}(x) \star Own_{\top}(x)\}$$

valid, provable

vacuous

Example

distributed counter

Let $p_1 \otimes q_1 = p_2 \otimes q_2 = \top$

$$\Gamma = r: \mathrm{Own}_{\top}(x) \star \mathrm{Own}_{p_1}(x_1) \star \mathrm{Own}_{p_2}(x_2) \wedge x=x_1+x_2$$

$\Gamma \vdash_{vr} \{\mathrm{Own}_{q_1}(x_1) \star \mathrm{Own}_{q_2}(x_2)\}$

with r **do** (x:=x+1; x_1:=x_1+1)

|| **with** r **do** (x:=x+1; x_2:=x_2+1)

$\{\mathrm{Own}_{q_1}(x_1) \star \mathrm{Own}_{q_2}(x_2)\}$

by

PARALLEL,
REGION

Permission transfer

The logic allows proofs in which *permissions* transfer implicitly between processes and resources

- For available resources, invariants hold separately
- Processes and resources maintain *compatible* permissions
- On *acquiring*, process *assumes* invariant, *claims* permissions
- At *release*, process *guarantees* invariant, *cedes* permissions

(cf. ownership transfer)

Validity

$\Gamma \vdash \{\varphi\}\ c\ \{\psi\}$ is *valid* iff...

Every finite computation of c
in an environment that respects Γ,
from a state satisfying $\varphi \star inv(\Gamma)$,
respects Γ, is race-free,
and ends in a state satisfying $\psi \star inv(\Gamma)$

(state = store + heap, with permissions)

Validity

DEFINITION

$\Gamma \vdash_{vr} \{\varphi\}\ c\ \{\psi\}$ is *valid* iff

For all $\alpha \in [\![c]\!]$, $\forall \sigma, \sigma'$.

if $\sigma \vDash \varphi$ and $\sigma \overset{\alpha}{\underset{\Gamma}{\Rightarrow}} \sigma'$

then $\sigma' \vDash \psi$

interactive computation in environment that respects Γ

(formalization of earlier definition)

Logical enabling

$$(\sigma, A) \overset{\alpha}{\underset{\Gamma}{\Rrightarrow}} (\sigma', A')$$

- When a process with resources A, in *local state* σ, can do α

σ is piece of global state claimed by process

- Assumes environment that respects Γ
- Causes abort if α violates permissions, breaks an invariant, or produces runtime error

... models permission transfer...

Logical enabling

READ

$$(\sigma, A) \overset{x=v}{\underset{\Gamma}{\Rrightarrow}} (\sigma, A) \quad \text{if } \exists p.\ \sigma(x) = (v, p)$$

$$(\sigma, A) \overset{x=v}{\underset{\Gamma}{\Rrightarrow}} \text{abort} \quad \text{if } x \notin dom(\sigma)$$

reading requires some permission

WRITE

$$(\sigma, A) \overset{x:=v}{\underset{\Gamma}{\Rrightarrow}} ([\sigma|x{:}(v,\top)], A) \quad \text{if } \exists v_0.\ \sigma(x) = (v_0, \quad)$$

$$(\sigma, A) \overset{x:=v}{\underset{\Gamma}{\Rrightarrow}} \text{abort} \quad \text{otherwise}$$

writing requires total permission

Logical enabling

ACQUIRE

$$(\sigma, A) \overset{acq(r)}{\underset{\Gamma}{\Rightarrow}} (\sigma \star \sigma', A \cup \{r\})$$

if $r \notin A,\ r{:}\vartheta \in \Gamma,\ \sigma \# \sigma',\ \sigma' \vDash \vartheta$

assume invariant; claim permissions

RELEASE

$$(\sigma, A) \overset{rel(r)}{\underset{\Gamma}{\Rightarrow}} (\sigma_1, A - \{r\})$$

if $r \in A,\ r{:}\vartheta \in \Gamma,\ \sigma = \sigma_1 \star \sigma_2,\ \sigma_2 \vDash \vartheta$

guarantee invariant; cede permissions

$$(\sigma, A) \overset{rel(r)}{\underset{\Gamma}{\Rightarrow}} abort$$

if $r \in A,\ r{:}\vartheta \in \Gamma,$

$\forall \sigma_1 \# \sigma_2.\ (\sigma = \sigma_1 \star \sigma_2$ *implies* $\sigma_2 \vDash \neg\vartheta)$

error if invariant fails

Soundness

THEOREM

Every provable formula is valid

PROOF

- Each inference rule preserves validity
- Key lemma: **PARALLEL DECOMPOSITION**

logical computations are compositional...

Parallel decomposition

LEMMA

Let $\alpha \in \alpha_1 || \alpha_2$ and $\sigma = \sigma_1 \star \sigma_2$

If $\sigma \overset{\alpha}{\underset{\Gamma}{\Rightarrow}}$ abort then $\sigma_1 \overset{\alpha_1}{\underset{\Gamma}{\Rightarrow}}$ abort or $\sigma_2 \overset{\alpha_2}{\underset{\Gamma}{\Rightarrow}}$ abort

If $\sigma \overset{\alpha}{\underset{\Gamma}{\Rightarrow}} \sigma'$ then $\sigma_1 \overset{\alpha_1}{\underset{\Gamma}{\Rightarrow}}$ abort

or $\sigma_2 \overset{\alpha_2}{\underset{\Gamma}{\Rightarrow}}$ abort

or $\exists\, \sigma_1', \sigma_2'.\ \sigma' = \sigma_1' \star \sigma_2'$ &

$\sigma_1 \overset{\alpha_1}{\underset{\Gamma}{\Rightarrow}} \sigma_1'$ & $\sigma_2 \overset{\alpha_2}{\underset{\Gamma}{\Rightarrow}} \sigma_2'$

Race-freedom

THEOREM

Validity of $\Gamma \vdash_{vr} \{\varphi\}\ c\ \{\psi\}$ implies

For all $\alpha \in [\![c]\!]$, $\forall \sigma, \sigma'$.

if $\sigma \vDash \varphi \star inv(\Gamma)$ and $\sigma \overset{\alpha}{\Rightarrow} \sigma'$

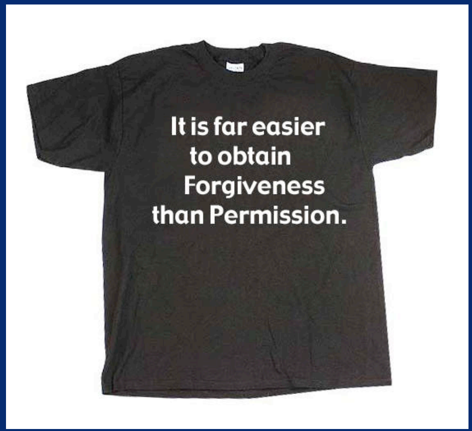
then $\sigma' \vDash \psi \star inv(\Gamma)$

interference-free global computation

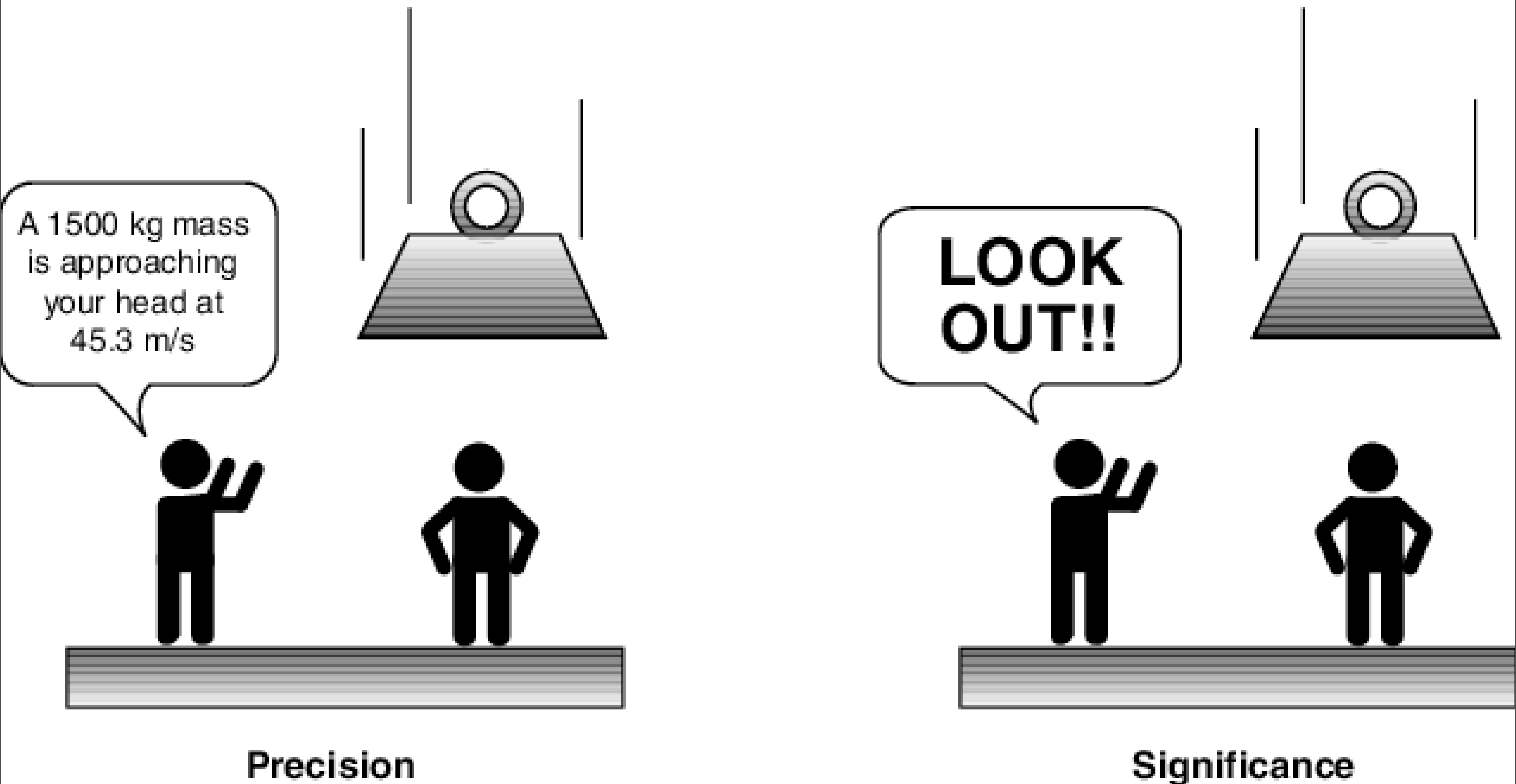
... NO RACES

Conclusions

- Concurrent permissions logic is sound
  - fractional, counting, ...
  - degenerate case $\mathcal{P} = \{\top\}$
- Evolution from earlier logics
  - general enough to handle *readers/writers*
  - uniform treatment of store and heap
- Significance of *precision*



Precision and Significance in the Real World



References

- Parkinson, Bornat and Calcagno ’06
  Variables as resource in Hoare logics, LICS 2006
- Brookes ’04
  A semantics for concurrent separation logic, CONCUR 2004
- O’Hearn ’04
  Resources, concurrency, and local reasoning, CONCUR 2004
- Reynolds ’02
  Separation logic: a logic for shared mutable data structures, LICS 2002