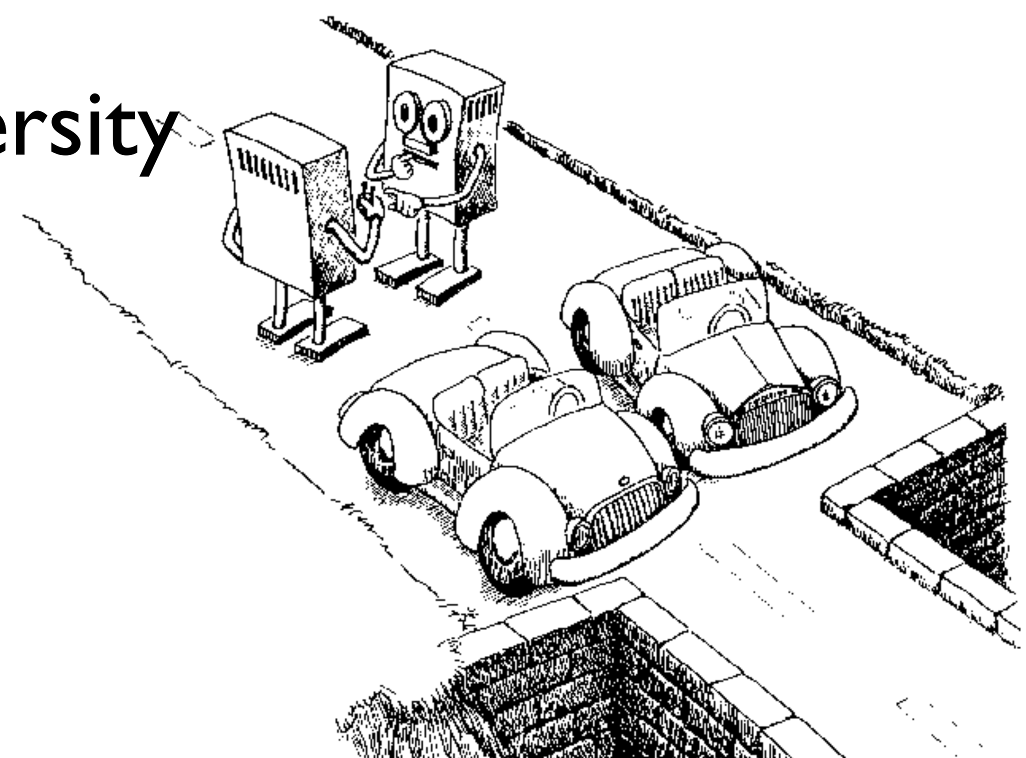
A race-detecting semantics for concurrent programs

Stephen Brookes
Carnegie Mellon University



Concurrency

- processes read and write to shared state
  - synchronize via conditional critical regions
  - mutually exclusive use of resources

Race condition

- Concurrent write to identifier being read or written by another process

$$x{:=}1 \parallel x{:=}2$$

What's the value of x?

Depends on granularity...

Solution?

- Ignore races semantically
- Assume known granularity

$$[\![x{:=}1 \parallel x{:=}2]\!] = \{x{:=}1\ x{:=}2,\ x{:=}2\ x{:=}1\}$$

High-level $x \in \{1, 2\}$

$$[\![x{:=}1 \parallel x{:=}2]\!] = (x.0{:=}1\ x.1{:=}0) \parallel (x.0{:=}0\ x.1{:=}1)$$

Low-level $x \in \{0, 1, 2, 3\}$

Solution?

- Avoid races syntactically
- Rules for critical variables *Owicki-Gries*
- Semantics just for race-free programs

Solution?

- Avoid races syntactically
- Rules for critical variables *Owicki-Gries*
- Semantics just for race-free programs

$x{:=}1 \parallel x{:=}2$ *disqualified*

Solution?

- Avoid races syntactically
- Rules for critical variables *Owicki-Gries*
- Semantics just for race-free programs

$x{:=}1 \parallel x{:=}2$ disqualified

$\mathbf{with}\ r\ \mathbf{do}\ x{:=}1 \parallel \mathbf{with}\ r\ \mathbf{do}\ x{:=}2$

$x \in \{1, 2\}$

Problems

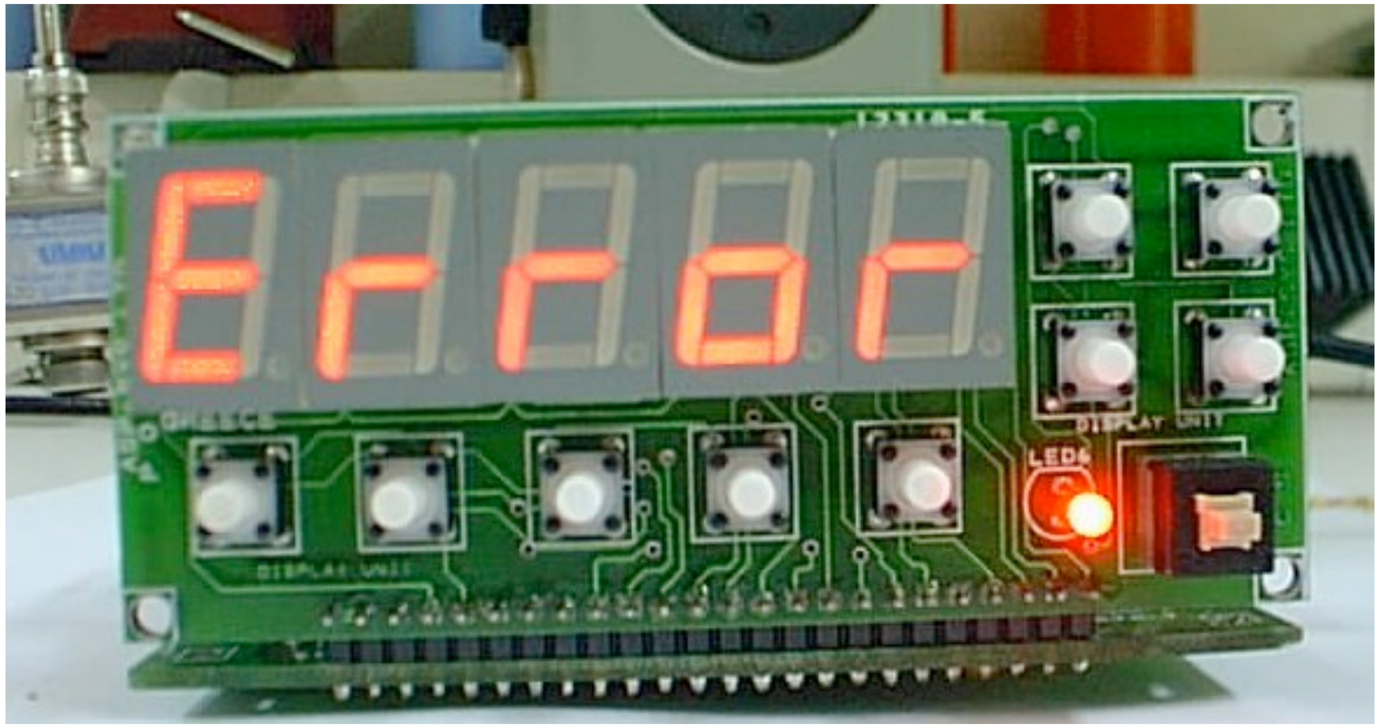
with the traditional approaches

- Granular semantics
  - combinatorial explosion
  - too specific, not uniform
- Race-avoiding syntax
  - static constraints have limits
  - too draconian

Solution

- Semantics with race-detection
  - potential race treated as catastrophic

cf. Reynolds



Outline

- Trace semantics
  - process denotes set of action traces
- High–level model
  - granularity of integer operations
- Low–level semantics
  - granularity of word operations
- Granularity Theorem
  - high-level consistent with low-level

Actions

High-level model

- idle λ
- read δ
- write $i{=}v$
- resource $i{:=}v$
- error $try(r), acq(r), rel(r)$

$abort$

$i \in$	**Ide**	identifiers
$r \in$	**Res**	resource names
$v \in$	V	integers

Traces

- Sequences of actions
- Concatenation

$$\alpha\delta\beta = \alpha\beta$$

$$\alpha \; \mathit{abort} \; \beta = \alpha \; \mathit{abort}$$

$$\alpha, \beta \in \mathbf{Tr}$$

State

- Global store s
  - maps identifiers to integers
- Resources
  - each process owns a finite set A
  - must be disjoint

changes dynamically...

Effects

- State ***enables*** certain actions
- Action has an effect

$$\boxed{(s,A) \overset{\lambda}{\Rightarrow} (s',A')}$$

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

Effects

$$(s, A) \stackrel{\delta}{\Longrightarrow} (s, A)$$

$$(s, A) \stackrel{i=v}{\Longrightarrow} (s, A)$$

if $(i,v) \in s$

$$(s, A) \stackrel{i:=v}{\Longrightarrow} ([s/i{:}v], A)$$

if $i \in dom(s)$

Effects

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

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

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

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

Effects

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

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

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

Semantics

of expressions

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

$$[\![i]\!] = \{(i{=}v, v) \mid v \in \mathrm{V}\}$$

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

Semantics
of commands

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

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

$$[\![c\ ;\ c']\!] = [\![c]\!]\ [\![c']\!]$$

$$[\![c \parallel c']\!] = [\![c]\!]\ {}_{\varnothing}\!\parallel_{\varnothing}\ [\![c']\!]$$

mutex fairmerge
with race-detection

Mutual exclusion

- At most one process holds each

$(acq(r)\ x{:=}1\ rel(r)) \mid\mid (acq(r)\ x{:=}2\ rel(r))$

only includes

$acq(r)\ x{:=}1\ rel(r)\ acq(r)\ x{:=}2\ rel(r)$

and

$acq(r)\ x{:=}2\ rel(r)\ acq(r)\ x{:=}1\ rel(r)$

Mutex fairmerge

Traditional Definition

$$(\lambda_1\,\alpha)_{A_1}\|_{A_2}(\lambda_2\,\beta)$$

$$= \quad \{\lambda_1\,\gamma \mid (A_1, A_2) \xrightarrow{\lambda_1} (A_1', A_2) \;\&\; \gamma \in \alpha_{A_1'}\|_{A_2}(\lambda_2\,\beta)\}$$

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

each process constrained by the other

to maintain disjoint sets of resources

Race-detecting mutex fairmerge

Definition

$$(\lambda_1\,\alpha)_{A_1}\|_{A_2}(\lambda_2\,\beta) = \{abort\}$$

if λ_1 *and* λ_2 *interfere*

$$= \quad \{\lambda_1\,\gamma \mid (A_1, A_2) \xrightarrow{\lambda_1} (A_1', A_2)\ \&\ \gamma \in \alpha_{A_1'}\|_{A_2}(\lambda_2\,\beta)\}$$
$$\cup\ \{\lambda_2\,\gamma \mid (A_2, A_1) \xrightarrow{\lambda_2} (A_2', A_1)\ \&\ \gamma \in (\lambda_1\alpha)_{A_1}\|_{A_2'}\beta\}$$

otherwise

Interference

Definition

$$\lambda_1 \bowtie \lambda_2$$

if

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

or

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

concurrent write to identifier being used by other process

Semantics

$$\boxed{[\![c \parallel c']\!] = [\![c]\!] \ {}_{\emptyset}\!\parallel_{\emptyset} [\![c']\!]}$$

mutex fairmerge
with
race detection

Example

$$[\![x := x+1 \;//\; x := x+1]\!]$$

$$= \{ x{=}v\ abort \mid v \in V \}$$

Semantics

$$[\![\textbf{with } r \textbf{ when } b \textbf{ do } c]\!] =$$

$$wait^{*}\ enter \ \cup\ wait^{\omega}$$

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

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

critical region protected by resource

Semantics

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

$[\![c]\!]_r$ *traces sequential for* r

$\alpha \backslash r$ *hide actions on* r

statically scoped local resource

Examples

$[\![\textbf{with } r \textbf{ do } x := x+1]\!] =$

$$try(r)^{\infty} \{acq(r)\ x{=}v\ x{:=}v{+}1\ rel(r) \mid v \in \mathrm{V}\}$$

$[\![\textbf{resource } r \textbf{ in } (\ldots) \parallel (\ldots)]\!] =$

$$[\![x := x+1 \,;\, x := x+1]\!]$$

Respect for resources

Lemma

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

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

then $A' = \emptyset$

Race-free programs

Definition

c is race-free from s

iff

$$\neg\, \exists \alpha \in [\![c]\!].\ \ s \overset{\alpha}{\Longrightarrow} \textbf{abort}$$

Low–level model

- Word size M
- Integer represented as *list of words*
- Word-level actions

$$\begin{array}{l} i.len=n \\ i.len:=n \\ i.0=w, \ldots, i.n=w \\ i.0:=w, \ldots, i.n:=w \end{array}$$

$0 \leq w < 2^M$

Low-level states

- Global store s
  - maps identifiers to *lists of words*
- Each process has set of resources A
  - pairwise disjoint

Low-level semantics

of expressions

$$[\![i]\!] =$$

$$\{(i.len{=}n\ i.0{=}w_0 \ldots i.n{=}w_n,\ [w_0 ,\ldots ,w_n]) \mid$$

$$n \geq 0\ \&\ w_0 ,\ldots ,w_n \in \mathbf{W}\ \}$$

$$[\![e+e']\!] = \{(\rho\rho', L \oplus L') \mid$$

$$(\rho, L) \in [\![e]\!]\ \&\ (\rho', L') \in [\![e']\!]\}$$

word-level arithmetic

Low-level semantics

of commands

$$[\![i{:=}e]\!] = \{\rho\ i.len{:=}n\ i.0{:=}w_0 \ldots i.n{:=}w_n \mid (\rho,[w_0,\ldots,w_n]) \in [\![e]\!]\}$$

Interference

at low level

$$writes(i.j{:=}v) \;=\; \{i.j\}$$
$$writes(i.len{:=}n) \;=\; \{\}$$
$$writes(i.j{=}v) = \{\}$$

$$reads(i.j{:=}v) \;=\; \{\}$$
$$reads(i.len{:=}n) \;=\; \{\}$$
$$reads(i.j{=}v) \;=\; \{i.j\}$$

Representation

Definitions

- Word lists represent integers

$$[w_0,w_1,\ldots,w_n]_M = w_0 + 2^M w_1 + \ldots + 2^{nM} w_n$$

- Low-level states represent high-level states

$$s_{low} \approx s_{high}$$

iff

$$[s_{low}(i)]_M = s_{high}(i)$$

for all identifiers i

Granularity Theorem (I)

- Every high-level error-free computation simulates a low-level computation

For all $\alpha \in [\![c]\!]_{high}$, s_{high} , s'_{high}, s_{low}

$$\begin{array}{ccc} s_{high} & \xRightarrow{\alpha} & s'_{high} \\ \approx & & \\ s_{low} & & \end{array}$$

IMPLIES

$$\begin{array}{ccc} & & s'_{high} \\ & & \approx \\ s_{low} & \xRightarrow{\beta} & s'_{low} \end{array}$$

for some $\beta \in [\![c]\!]_{low}$, s'_{low}

Granularity Theorem(2)

- Every low-level computation is (weakly) simulated by a high-level computation

For all $\beta \in [\![c]\!]_{low}$, s_{high} , s'_{low}, s_{low}

$$\begin{tikzcd} s_{high} \arrow[d, phantom, "\wr\wr"] & \\ s_{low} \arrow[r, Rightarrow, "\beta"] & s'_{low} \end{tikzcd}$$

IMPLIES

$$\begin{tikzcd} s_{high} \arrow[r, Rightarrow, "\alpha"] & s'_{high} \arrow[d, phantom, "\wr\wr"] \\ & s'_{low} \end{tikzcd}$$

OR

$$s_{high} \overset{\alpha}{\Longrightarrow} \boldsymbol{abort}$$

for some $\alpha \in [\![c]\!]_{high}$, s'_{high}

Race–free case

- Simulation both ways if c is race-free

$$\begin{array}{ccc} s_{high} & & \\ \wr & & \\ s_{low} & \overset{\beta}{\Longrightarrow} & s'_{low} \end{array}$$

IFF

$$\begin{array}{ccc} s_{high} & \overset{\alpha}{\Longrightarrow} & s'_{high} \\ & & \wr \\ & & s'_{low} \end{array}$$

More concrete

- Low-level state = store + heap
- Effect of *i.len:=n* includes *allocation* and/or *deallocation* of heap cells
- Results still go through

Further work

- Soundness of Owicki-Gries logic
- Ideas extend to include pointers
  - concurrent separation logic

    O'Hearn, Brookes

    to appear at

    CONCUR '04