A grainless semantics for concurrent separation logic

Stephen Brookes
CMU

Géométrie du Calcul
GEOCAL '06

Shared-memory programs

- Parallel composition

  $c_1 \mid\mid c_2$

- Conditional critical region

  with r **when** b **do** c

- Local resource declaration

  resource r **in** c

Traditional semantics

- Program denotes a set of *traces*
- Trace = sequence of *atomic actions*
- Parallel composition = *fair interleaving*
- Resource acquisition is *mutually exclusive*

Granularity

Traditional models assume a *default level of granularity* for *atomic actions*

- integer assignments — *coarse*
- reads and writes to integer variables — *fine*
- reads and writes to machine words — *word-level*

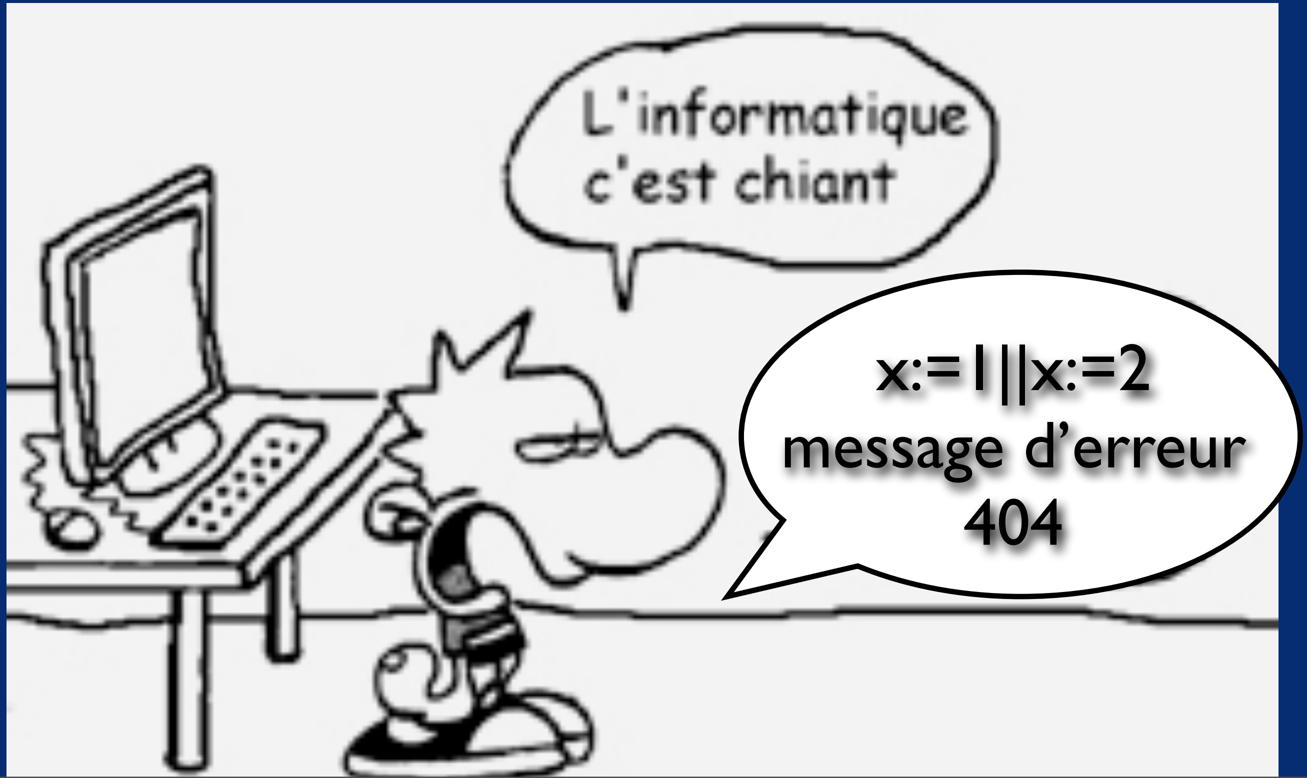
Example

x := x+1 || x := x+2

- coarse *x increases by 3*
- fine *x increases by 1, 2, or 3*
- word *depends on word size*

Danger

A *concurrent write*
to a variable being used by another process ...



Danger

A *concurrent write*
to a variable being used by another process ...

- race condition
- unpredictable results

Status quo

- Traditional models *ignore* race conditions
  by assuming that atomic actions never overlap
- Each supports compositional reasoning
  for *race-free* programs
- But they each interpret *racy* programs differently

Race-freedom

- Partial correctness behavior of a *race-free* program is *independent* of granularity

Should be able to abstract away from granularity

... but traditional models don't do this!

The Dijkstra Principle

Similarly...

... processes should be *loosely connected*;
... apart from the (rare) moments of explicit communication, processes are to be regarded as *completely independent* of each other

The Dijkstra Principle

That is...

... should be able to abstract away from what happens between synchronizations

The Dijkstra Principle

That is...

... should be able to abstract away from what happens between synchronizations

WARNING
Not reflected in design of traditional models

Traditional logic

... based on Dijkstra's Principle

Owicki/Gries '76

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

- Resource-sensitive partial correctness
  - Γ specifies *resources* r_i, *protection lists* X_i, and *invariants* R_i
  - p, q describe *unprotected* variables
- Static constraints guarantee race-freedom
  - *critical variables* must be protected
  - protected variables only allowed inside region

Parallel rule

Owicki/Gries

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

provided

$$\mathrm{free}(p_1, q_1) \cap \mathrm{writes}(c_2) = \varnothing$$
$$\mathrm{free}(p_2, q_2) \cap \mathrm{writes}(c_1) = \varnothing$$
$$\mathrm{free}(c_1) \cap \mathrm{writes}(c_2) \subseteq \mathrm{owned}(\Gamma)$$
$$\mathrm{free}(c_2) \cap \mathrm{writes}(c_1) \subseteq \mathrm{owned}(\Gamma)$$

critical variables are protected

Resource rules

Owicki/Gries

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

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

(subject to well-formedness conditions)

Validity

Definition

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

Every finite computation of c
in an environment that respects Γ,
from $p \wedge R_1 \wedge \ldots \wedge R_n$,
respects Γ, is race-free,
and ends in $q \wedge R_1 \wedge \ldots \wedge R_n$

(made formal using a traditional model)

Soundness
Theorem

- Owicki-Gries logic is *sound* for pointer-less programs
  - Every provable formula is *valid*

Can use *any* of the
traditional semantic models...

Compositionality

Theorem

Traditional traces support compositional reasoning

- Let c_1 and c_2 be code fragments that denote the *same trace set*
- Let $C[-]$ be a program context
- If $\Gamma \vdash \{p\}\ C[c_1]\ \{q\}$ is valid, so is $\Gamma \vdash \{p\}\ C[c_2]\ \{q\}$

same traces

$\Rightarrow$

same behavior, in all contexts

Semantic problems

Traditional models don't help much!

These models...

- make too many distinctions

$$[\![x{:=}x{+}1; x{:=}x{+}1]\!] \neq [\![x{:=}x{+}2]\!]$$

$$[\![x{:=}1; y{:=}2]\!] \neq [\![y{:=}2; x{:=}1]\!]$$

- do not reflect Dijkstra's principle

  ... contain unnecessary traces

- suffer from combinatorial explosion

  ... involve unnecessary interleavings

Logic problems

Traditional logic doesn't generalize...

- Owicki-Gries logic is *not sound for pointer programs*
- Static constraints cannot prevent pointer races, because of *aliasing*

[x]:=1 || [y]:=2 *races* if x and y are aliases	dispose x || dispose y *races* if x and y are aliases
concurrent update	*dangling pointer*

Pointer programs

- Lookup

  $i := [e]$

- Update

  $[e] := e'$

- Allocation

  $i :=$ **cons** $(e_1, \ldots, e_n)$

- Disposal

  dispose e

Reasoning about pointers

Reynolds '02

- Hoare-style rules for *sequential* pointer-programs
- State = store + heap
- Pre- and post-conditions drawn from *separation logic*
  - emp : heap is empty
  - $e \mapsto e'$: singleton heap
  - $p_1 \star p_2$: heap can be split so that p_1 and p_2 hold *separately*

A proposal

O'Hearn '02

- Combine Owicki-Gries with separation logic
  - Let resource invariants be *separation logic* formulas
  - Use ✶ strategically to prevent aliasing

an apparently simple idea
with deep consequences

Parallel rule

O'Hearn '02

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

provided

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

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

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

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

Parallel rule

O'Hearn '02

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

$\star$ for $\wedge$

provided

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

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

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

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

Parallel rule

O'Hearn '02

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

$\star$ for $\wedge$

provided

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

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

same as before

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

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

Resource rules

O'Hearn '02

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

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

Resource rules

O'Hearn '02

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

$\star$ for $\wedge$

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

Resource rules

O'Hearn '02

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

$\star$ for $\wedge$

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

$\star$ for $\wedge$

Validity

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

Every finite computation of c
in an environment that respects Γ,
from $p \star R_1 \star \ldots \star R_n$,
respects Γ, is race-free,
and ends in $q \star R_1 \star \ldots \star R_n$

An intuitive definition,
in need of formalization...

Ownership transfer

- The logic allows proofs in which *ownership* of a pointer *transfers* implicitly between processes and resources, based on resource invariants

  - for each *available* resource, invariant holds *in a sub-heap*
  - when *acquiring* a resource, process assumes invariant, *claims* ownership of the protected variables + sub-heap
  - when *releasing* a resource, process guarantees that invariant holds in some sub-heap, *cedes* ownership

Example

1-place shared buffer

PUT :: **with** buf **when** full=0 **do** (z := x; full := 1)

GET :: **with** buf **when** full=1 **do** (y := z; full := 0)

Let Γ = buf(z,full): $(\mathrm{full}=1 \wedge z \mapsto \_\ ) \vee (\mathrm{full}=0 \wedge \mathrm{emp})$

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

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

ownership
passes
from left to right

$\Gamma \vdash \{x \mapsto \_\}\ \mathrm{PUT} \parallel (\mathrm{GET};\ \mathrm{dispose}\ y)\ \{\mathrm{emp}\}$

Example, take 2

1-place shared buffer

Using a different invariant...

$\Gamma' = \text{buf(z,full)}: (\text{full}=1 \wedge \text{emp}) \vee (\text{full}=0 \wedge \text{emp})$

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

$\Gamma' \vdash \{\text{emp}\}\ \text{GET}\ \{\text{emp}\}$

no transfer
of ownership

$\Gamma' \vdash \{x \mapsto \_\}\ (\text{PUT; dispose } x) \parallel \text{GET}\ \{\text{emp}\}$

Example, take 3

1-place shared buffer

And we cannot prove a racy program...

dangling pointer

$\Gamma \vdash \{x \mapsto \_\}$ (PUT; dispose x) || (GET; dispose y) {...}

not provable, for any Γ

ownership cannot go both ways!

Soundness?

- Far from obvious! Ownership is a tricky concept...
  - cannot rely on Owicki/Gries
  - resource invariants must be restricted (Reynolds '02)
- Need a semantics
  - must account for ownership transfer
  - ideally, should be *grainless*
- Traditional models won't work...

A new semantics

based on Dijkstra's principle

Footstep traces

- No interference except on synchronization
  - *built into structure of traces*
- Treats race condition as *disaster*
  - *built into definition of interleaving*
- Independent of granularity
  - *abstracts away from state changes between synchronizations*

Advantages

- Succinctness

  big steps, fewer traces, fewer interleavings

- Simplicity

  supports "sequential" reasoning
  for synchronization-free code

- Soundness

  can be used to prove soundness of
  concurrent separation logic

States

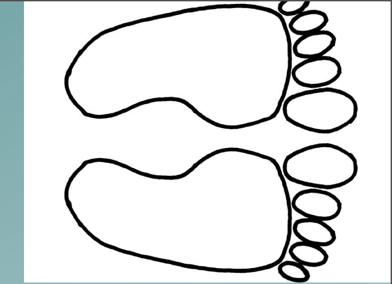
Definition

A state specifies values for identifiers and heap cells

- $St = Var \rightharpoonup V_{int}$ *states*
- $Var = Ide \uplus Loc$ *variables*
- $Loc \subseteq V_{int}$ *heap cells*

Let σ range over the set of states...

Footsteps



A *footstep* $(\sigma, \sigma')_X$ describes the *footprint* of a sequence of state changes:

change σ to σ',
while only *reading* the variables in X

$(\sigma, \sigma')_X$

Resource actions

- *try*(r) unsuccessful attempt
- *acq*(r) acquisition
- *rel*(r) release

Used to model synchronization

Traces

loosely connected
sequences of actions

- Built from *catenable* actions

$$([x{:}0], [x{:}1])_{\emptyset} \asymp ([x{:}1, y{:}0], [x{:}1, y{:}1])_{\{x\}}$$

- Consecutive footsteps get *stumbled* together

$$([x{:}0], [x{:}1])_{\emptyset} \; ; ([x{:}1, y{:}0], [x{:}1, y{:}1])_{\{x\}} = ([x{:}0, y{:}0], [x{:}1, y{:}1])_{\emptyset}$$

$$([v{:}0], [\ ])_{\emptyset} \; ; ([v{:}0], [\ ])_{\emptyset} = ([v{:}0], \textit{abort})$$

- Interference only on synchronization

$$(\sigma_0, \sigma_0')_{X_0} \; \mathrm{acq}(r) \; (\sigma_1, \sigma_1')_{X_1} \; \mathrm{rel}(r) \; (\sigma_2, \sigma_2')_{X_2} \; \ldots$$

Semantics

- A command denotes a set of *traces*

$$[\![c]\!] \subseteq Tr$$

- Defined by structural induction on c

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

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

resource-sensitive, race-detecting,
fair interleaving

Examples

$[\![x{:=}1;\ y{:=}2]\!] = [\![y{:=}2;\ x{:=}1]\!]$

$= \{\ ([x{:}v, y{:}v'], [x{:}1, y{:}2])_{\emptyset} \mid v, v' \in V_{int}\ \}$

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

$= \{\ ([x{:}v], \mathit{abort}) \mid v \in V_{int}\ \}$

$[\![$ resource r in

with r do x:=x+1 || with r do x:=x+2 $]\!]$

$= \{\ ([x{:}v], [x{:}v{+}3])_{\emptyset} \mid v \in V_{int}\ \}$

Theorem

Footstep traces
allow sequential reasoning
for resource-free programs

Every trace of a *resource-free program*
is a single footstep

c is resource-free if $res(c)=\varnothing$ (no free resource names)

Every sequential program is resource-free

Race-free

Definition

c is *race-free from* σ

iff

$\forall \alpha \in [\![c]\!].\ \neg\,(\sigma \overset{\alpha}{\Rightarrow} \mathit{abort})$

no trace leads to an error

Example

full := 0; **resource** buf **in**
(x := **cons**(-); PUT; **dispose** x) || GET

race-free

has footstep traces

([full:_, x:_, y:_, z:_], [full:0, x:v, y:v, z:v])$_{\varnothing}$

Example

full := 0;
resource buf **in**
(x := **cons**(-); PUT) || (GET; **dispose** y)

ownership transfer

race-free

... has the same trace set!

(not true in traditional models)

Explosion?

N^2 2^N $N!$

N puts,
N gets

full := 0;
resource buf **in**
    (x := **cons**(-); PUT)^N || (GET; **dispose** y)^N

independent of N

... has the same trace set!

(not true in traditional models)

Racy example

full := 0; x := **cons**(-);
resource buf **in**
(PUT; **dispose** x) || (GET; **dispose** y)

has traces

([full:_, x:_, y:_, z:_], *abort*)

race condition modelled correctly

Soundness
Theorem

Brookes '04

- Every provable formula of concurrent separation logic is *valid*, provided resource invariants are *precise*
  - Footstep traces permit rigorous treatment of *ownership transfer*
  - Show that each inference rule preserves validity
    - proof reveals crucial role of *precision*

⇒ *Every provable program is race-free*

Compositionality

Theorem

Footstep traces support compositional reasoning for parallel pointer-programs

- Let c_1 and c_2 be code fragments with the same *footstep traces*
- Let $C[-]$ be a program context
- If $\Gamma \vdash \{p\}C[c_1]\{q\}$ is valid, so is $\Gamma \vdash \{p\}C[c_2]\{q\}$

same traces

$\Rightarrow$

same behavior, in all contexts

Advantages

significant help for compositional reasoning!

Footstep trace semantics...

- makes fewer distinctions

$$[\![x{:=}x{+}1; x{:=}x{+}1]\!] = [\![x{:=}x{+}2]\!]$$

$$[\![x{:=}1; y{:=}2]\!] = [\![y{:=}2; x{:=}1]\!]$$

- embodies Dijkstra's principle

  ... *only loosely connected traces*

- ameliorates the combinatorial explosion

  ... *fewer interleavings*

Conclusions

- Semantics abstracts away from inessential details between synchronizations
  - *facilitates reasoning about loosely connected processes*
- Logic provides safe reasoning about concurrency + pointers
  - *every provable program is race-free*

Ideas should be more widely applicable