A grainless semantics for concurrent separation logic

Stephen Brookes
CMU

Géométrie du Calcul
GEOCAL ‘06
Shared-memory programs

- Parallel composition
  \[ c_1 \parallel c_2 \]

- Conditional critical region
  \[ \text{with } r \text{ when } b \text{ do } c \]

- Local resource declaration
  \[ \text{resource } r \text{ in } c \]
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

\[ x := x + 1 \quad \text{||} \quad x := x + 2 \]

- coarse: \( x \) increases by 3
- fine: \( x \) increases by 1, 2, or 3
- word: depends on word size
A concurrent write to a variable being used by another process ...

\[ x := 1 \| x := 2 \]

L'informatique c'est chiant

message d'erreur 404
A concurrent write to a variable being used by another process...

- race condition
- unpredictable results
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

Resource-sensitive partial correctness

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

- $\Gamma$ 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

Owicki/Gries ‘76
Parallel rule

\[ \Gamma \vdash \{p_1\} c_1 \{q_1\} \quad \Gamma \vdash \{p_2\} c_2 \{q_2\} \]

\[ \Gamma \vdash \{p_1 \land p_2\} c_1 \parallel c_2 \{q_1 \land q_2\} \]

provided

\[
\begin{align*}
\text{free}(p_1,q_1) \cap \text{writes}(c_2) &= \emptyset \\
\text{free}(p_2,q_2) \cap \text{writes}(c_1) &= \emptyset \\
\text{free}(c_1) \cap \text{writes}(c_2) &\subseteq \text{owned}(\Gamma) \\
\text{free}(c_2) \cap \text{writes}(c_1) &\subseteq \text{owned}(\Gamma)
\end{align*}
\]
Resource rules

\[ \Gamma \vdash \{(p \land R) \land b\} \ c \ \{q \land R\} \]

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

\[ \Gamma, r(X):R \vdash \{p\} \ c \ \{q\} \]

\[ \Gamma \vdash \{p \land R\} \textbf{resource} \ r \ \textbf{in} \ c \ \{q \land R\} \]

(subject to well-formedness conditions)
\( \Gamma \vdash \{p\}c\{q\} \) is valid if:

Every finite computation of \( c \) in an environment that respects \( \Gamma \), from \( p \land R_1 \land \ldots \land R_n \), respects \( \Gamma \), is race-free, and ends in \( q \land R_1 \land \ldots \land R_n \).

*(made formal using a traditional model)*
Owicki-Gries logic is \textit{sound} for pointer-less programs

Every provable formula is \textit{valid}

Can use \textit{any} of the traditional semantic models...
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  
$\Rightarrow$

dsane behavior, in all contexts
Semantic problems

These models...

- make too many distinctions

\[
\begin{align*}
&[x:=x+1; x:=x+1] 
\neq
\begin{cases}
[x:=x+2] \\
[x:=1; y:=2]
\end{cases}
\neq
\begin{cases}
y:=2; x:=1
\end{cases}
\end{align*}
\]

- do not reflect Dijkstra’s principle
  - ... contain unnecessary traces
- suffer from combinatorial explosion
  - ... involve unnecessary interleavings

Traditional models don’t help much!
Owicki-Gries logic is **not sound for pointer programs**.

Static constraints cannot prevent pointer races, because of aliasing.

- Concurrent update: $[x] := 1 \parallel [y] := 2$ races if $x$ and $y$ are aliases.
- Dangling pointer: `dispose x \parallel dispose y` races if $x$ and $y$ are aliases.
Pointer programs

- **Lookup**
  
  \[ i := [e] \]

- **Update**
  
  \[ [e] := e' \]

- **Allocation**
  
  \[ i := \text{cons} \ (e_1, \ldots, e_n) \]

- **Disposal**
  
  \[ \text{dispose} \ e \]
Reasoning about pointers

- Hoare-style rules for *sequential* pointer-programs

- State = store + heap

- Pre- and post-conditions drawn from *separation* logic

  - $\text{emp}$ : 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

Reynolds ’02
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

\[
\Gamma \vdash \{p_1\} c_1 \{q_1\} \quad \Gamma \vdash \{p_2\} c_2 \{q_2\}
\]

\[
\Gamma \vdash \{p_1 \star p_2\} c_1 \parallel c_2 \{q_1 \star q_2\}
\]

provided

\[
\text{free}(p_1, q_1) \cap \text{writes}(c_2) = \emptyset
\]

\[
\text{free}(p_2, q_2) \cap \text{writes}(c_1) = \emptyset
\]

\[
\text{free}(c_1) \cap \text{writes}(c_2) \subseteq \text{owned}(\Gamma)
\]

\[
\text{free}(c_2) \cap \text{writes}(c_1) \subseteq \text{owned}(\Gamma)
\]
Parallel rule

\[ \Gamma \vdash \{p_1\} c_1 \{q_1\} \quad \Gamma \vdash \{p_2\} c_2 \{q_2\} \]

\[ \Gamma \vdash \{p_1 \star p_2\} c_1 || c_2 \{q_1 \star q_2\} \]

provided

\[\text{free}(p_1,q_1) \cap \text{writes}(c_2) = \emptyset\]
\[\text{free}(p_2,q_2) \cap \text{writes}(c_1) = \emptyset\]
\[\text{free}(c_1) \cap \text{writes}(c_2) \subseteq \text{owned}(\Gamma)\]
\[\text{free}(c_2) \cap \text{writes}(c_1) \subseteq \text{owned}(\Gamma)\]

for \(\wedge\)
Parallel rule

\[\Gamma \vdash \{p_1\} c_1 \{q_1\} \quad \Gamma \vdash \{p_2\} c_2 \{q_2\}\]

\[\Gamma \vdash \{p_1 \star p_2\} c_1 || c_2 \{q_1 \star q_2\}\]

provided

\[\text{free}(p_1, q_1) \cap \text{writes}(c_2) = \emptyset\]
\[\text{free}(p_2, q_2) \cap \text{writes}(c_1) = \emptyset\]
\[\text{free}(c_1) \cap \text{writes}(c_2) \subseteq \text{owned}(\Gamma)\]
\[\text{free}(c_2) \cap \text{writes}(c_1) \subseteq \text{owned}(\Gamma)\]

\[\star \text{ for } \land\]

same as before

O’Hearn ’02

for \land

\star

same as before
Resource rules

\[ \Gamma \vdash \{(p \star R) \land b\} \ c \ \{q \star R\} \]

\[ \Gamma, r(X):R \vdash \{p\} \textit{ with } r \textit{ when } b \textit{ do } c \ \{q\} \]

\[ \Gamma, r(X):R \vdash \{p\} \ c \ \{q\} \]

\[ \Gamma \vdash \{p \star R\} \textit{ resource } r \textit{ in } c \ \{q \star R\} \]
Resource rules

\[ \Gamma \vdash \{(p \star R) \land b\} \ c \ \{q \star R\} \]

\[ \Gamma, r(X):R \vdash \{p\} \ \text{with} \ r \ \text{when} \ b \ \text{do} \ c \ \{q\} \]

\[ \Gamma, r(X):R \vdash \{p\} \ c \ \{q\} \]

\[ \Gamma \vdash \{p \star R\} \ \text{resource} \ r \ \text{in} \ c \ \{q \star R\} \]
Resource rules

\[
\frac{\Gamma \vdash \{(p \succ R) \land b\} \ c \ \{q \succ R\}}{} \quad \text{\textit{\# for} } \land \\
\frac{\Gamma, r(X):R \vdash \{p\} \ \textit{with} \ r \ \textit{when} \ b \ \textit{do} \ c \ \{q\}}{} \quad \text{\textit{\# for} } \land \\
\frac{\Gamma, r(X):R \vdash \{p\} \ c \ \{q\}}{} \quad \text{\textit{\# for} } \land \\
\frac{\Gamma \vdash \{p \succ R\} \ \textit{resource} \ r \ \textit{in} \ c \ \{q \succ R\}}{} \quad \text{\textit{\# for} } \land
\]

O’Hearn ’02
\[ \Gamma \vdash \{p\}c\{q\} \text{ is valid if:} \]

Every finite computation of \( c \) in an environment that respects \( \Gamma \), from \( p \ast R_1 \ast \ldots \ast R_n \), respects \( \Gamma \), is race-free, and ends in \( q \ast R_1 \ast \ldots \ast R_n \)

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

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

Let \( \Gamma = \text{buf}(z,\text{full}):(\text{full}=1 \land z \mapsto \_ ) \lor (\text{full}=0 \land \text{emp}) \)

\[ \Gamma \vdash \{ x \mapsto \_ \} \text{ PUT } \{ \text{emp} \} \]
\[ \Gamma \vdash \{ \text{emp} \} \text{ GET } \{ y \mapsto \_ \} \]
\[ \Gamma \vdash \{ x \mapsto \_ \} \text{ PUT } \| (\text{GET}; \text{dispose y}) \{ \text{emp} \} \]
Using a different invariant...

\[ \Gamma' = \text{buf}(z, \text{full}): (\text{full} = 1 \land \text{emp}) \lor (\text{full} = 0 \land \text{emp}) \]

\[ \Gamma' \vdash \{x \mapsto \_\} \text{PUT} \{x \mapsto \_\} \]

\[ \Gamma' \vdash \{\text{emp}\} \text{GET} \{\text{emp}\} \]

\[ \Gamma' \vdash \{x \mapsto \_\} (\text{PUT}; \text{dispose } x) \parallel \text{GET} \{\text{emp}\} \]
And we cannot prove a racy program...

\[ \Gamma \vdash \{ x \mapsto _\_ \} (\text{PUT}; \text{dispose } x) \parallel (\text{GET}; \text{dispose } y) \{ \ldots \} \]

not provable, for any \( \Gamma \)

ownership cannot go both ways!
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
A state specifies values for identifiers and heap cells.

- \( St = Var \rightarrow V_{int} \)  
  - **States**

- \( Var = Ide \cup Loc \)  
  - **Variables**

- \( Loc \subseteq V_{int} \)  
  - **Heap cells**

Let \( \sigma \) range over the set of states...
A footprint \((\sigma, \sigma')_X\) describes the footprint of a sequence of state changes:

change \(\sigma\) to \(\sigma'\),
while only reading the variables in \(X\)
Resource actions

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

Used to model synchronization
Traces

- Built from **catenable** actions
  
  \((\([x:0], [x:1]\)]\)\(\emptyset\) \cong (\([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]\])\(\emptyset\)

  (\([v:0], [ ]\)]\(\emptyset\) ; (\([v:0], [ ]\)]\(\emptyset\) = (\([v:0], \text{abort}\)\)

- Interference only on **synchronization**

  \((\sigma_0, \sigma_0')_{x_0} \text{acq}(r) (\sigma_1, \sigma_1')_{x_1} \text{rel}(r) (\sigma_2, \sigma_2')_{x_2} ...\)
Semantics

A command denotes a set of traces
\[ \llbracket c \rrbracket \subseteq \text{Tr} \]

Defined by structural induction on \( c \)

\[ \llbracket c_1 ; c_2 \rrbracket = \{ \alpha_1 ; \alpha_2 \mid \alpha_1 \in \llbracket c_1 \rrbracket, \alpha_2 \in \llbracket c_2 \rrbracket, \alpha_1 \approx \alpha_2 \} \]

\[ \llbracket c_1 || c_2 \rrbracket = \bigcup \{ \alpha_1 || \alpha_2 \mid \alpha_1 \in \llbracket c_1 \rrbracket, \alpha_2 \in \llbracket c_2 \rrbracket \} \]

resource-sensitive, race-detecting, fair interleaving
Examples

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

\[
[[x:=x+1 \mid\mid x:=x+2]]
= \{ ([x:v], abort) \mid v \in V_{int} \}
\]

\[
[[\text{resource } r \text{ in}
  \text{with } r \text{ do } x:=x+1 \mid\mid \text{with } r \text{ do } x:=x+2 ]]
= \{ ([x:v], [x:v+3]) \_ | v \in V_{int} \}
\]
Theorem

Footstep traces allow sequential reasoning for resource-free programs

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

\[ c \text{ is resource-free if } \text{res}(c) = \emptyset \quad \text{(no free resource names)} \]

Every sequential program is resource-free
**Race-free**

**Definition**

**c is race-free from $\sigma$**

iff

$$\forall \alpha \in \llbracket c \rrbracket. \neg (\sigma \xrightarrow{\alpha} \text{abort})$$

no trace leads to an error
Example

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

has footstep traces

```
([full:_, x:_, y:_, z:_], [full:0, x:v, y:v, z:v])∅
```
Example

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

... has the same trace set!

(not true in traditional models)
Explosion?

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

... has the same trace set!

N puts, N gets

... 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
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.
Let $c_1$ and $c_2$ be code fragments with the same \textit{footstep traces}.

Let $C[-]$ be a program context.

If $\Gamma \vdash \{p\}C[c_1]\{q\}$ is valid, so is $\Gamma \vdash \{p\}C[c_2]\{q\}$.

\textit{same traces} $\Rightarrow$ \textit{same behavior, in all contexts}
Advantages

Footstep trace semantics...

- makes fewer distinctions
  \[
  \begin{align*}
  &\left[ x:=x+1; x:=x+1 \right] = \left[ x:=x+2 \right] \\
  &\left[ x:=1; y:=2 \right] = \left[ y:=2; x:=1 \right]
  \end{align*}
  \]
- embodies Dijkstra's principle
  ... only loosely connected traces
- ameliorates the combinatorial explosion
  ... fewer interleavings

significant help for compositional reasoning!
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