A Brief History of Shared Memory

Stephen Brookes
CMU
Outline

- Revisionist history
  - Rational reconstruction of early models
  - Evolution of recent models
- A unifying framework
  - Fault-detecting trace semantics
- Some general results
  - Soundness of fault-avoiding logics
Framework

- An abstract notion of state and action
- A recipe for constructing denotational models
  - sequential programs
  - shared memory parallel programs
- Designed to support compositional reasoning
  - fault-avoiding correctness
  - rely/guarantee properties
A state model is a tuple \((S, A, \rightarrow, \#)\) with

\[
S = (S, \otimes)
\]

\(A\)

\(\rightarrow \subseteq S \times A \times S^\dagger\)

\(S^\dagger = S \cup \{\text{error}\}\)

and

\[
\otimes : S \times S \rightarrow S
\]

\(\# \subseteq A \times A\)

satisfying natural axioms ...
**State axioms**

* $(S, \otimes)$ is a partial commutative monoid...

\[
\sigma \otimes \tau \simeq \tau \otimes \sigma
\]

\[
\rho \otimes (\sigma \otimes \tau) \simeq (\rho \otimes \sigma) \otimes \tau
\]

* ... with unique decomposition

\[
\sigma \otimes \sigma_1 = \sigma \otimes \sigma_2 \Rightarrow \sigma_1 = \sigma_2
\]
Footprint axioms

• **Successful action has unique cause**

For all $\sigma, \lambda$

at most one $\sigma_1$ such that

$\sigma_1 \xrightarrow{\lambda} \sigma_1'$, $\sigma = \sigma_1 \otimes \sigma_2$

• **Failure is irrevocable**

If $\sigma_1 \otimes \sigma_2 \xrightarrow{\lambda} \text{error}$,

then $\sigma_1 \xrightarrow{\lambda} \text{error}$
**Independence axioms**

* Independence implies non-interfering footprints

\[ \lambda_1 \not\# \lambda_2 \quad \& \quad \sigma_1 \xrightarrow{\lambda_1} \sigma_1' \quad \& \quad \sigma_2 \xrightarrow{\lambda_2} \sigma_2' \]
\[ \& \quad \sigma = \sigma_1 \otimes \tau_1 = \sigma_2 \otimes \tau_2 \]

implies

\[ \exists \tau_1', \tau_2'. \quad \sigma_1' \otimes \tau_1 = \sigma_2 \otimes \tau_2' \]
\[ \& \quad \sigma_2' \otimes \tau_2 = \sigma_1 \otimes \tau_1' \]
\[ \& \quad \sigma_1' \otimes \tau_1' = \sigma_2' \otimes \tau_2' \]

* Symmetry

\[ \lambda_1 \not\# \lambda_2 \text{ implies } \lambda_2 \not\# \lambda_1 \]
Enabling

For any state model we can derive an enabling relation

\[ \Rightarrow \subseteq S^\dagger \times A \times S^\dagger \]

Let $\sigma \xrightarrow{\lambda} \sigma'$ iff $\exists \sigma_1, \sigma_1', \sigma_2 \in S$.

\[ \sigma = \sigma_1 \times \sigma_2 \]
\[ \& \quad \sigma' = \sigma_1' \times \sigma_2 \]
\[ \& \quad \sigma_1 \xrightarrow{\lambda} \sigma_1' \]

Let $\sigma \xrightarrow{\lambda} \text{error}$ iff $\sigma \xrightarrow{\lambda} \text{error}$ or $\sigma = \text{error}$
**CONSEQUENCES**

- **Frame**

\[
\sigma_1 \xrightarrow{\lambda} T_1 \neq \text{error} \quad \& \quad \sigma_1 \times \sigma_2 \xrightarrow{\lambda} T
\]

implies

\[T = T_1 \times \sigma_2\]

- **Safety monotonicity**

\[
\sigma_1 \times \sigma_2 \xrightarrow{\lambda} \text{error}
\]

implies

\[\sigma_1 \xrightarrow{\lambda} \text{error}\]
Independent actions don’t interfere

If $\lambda_1 \neq \lambda_2$ then

$\sigma \Rightarrow \tau_1$, $\sigma \Rightarrow \tau_2$

implies

$\exists \tau. \quad \tau_1 \Rightarrow \tau$, $\tau_2 \Rightarrow \tau$
EXAMPLE

global transition traces

- $S = (\text{Ide} \rightarrow V) \cup \{1\}$
- $\sigma \otimes 1 = \sigma = 1 \otimes \sigma$
- $A = S \times S$
- $(\sigma, \tau)$
- $\sigma \rightarrow \tau$
- $(\sigma_1, \tau_1) \not\equiv (\sigma_2, \tau_2)$ iff $\sigma_1 = \tau_1 = \sigma_2 = \tau_2$

cf. Park 1979
EXAMPLE

local transition traces

$S = \text{Ide} \xrightarrow{\text{fin}} V$

- $\boxtimes$ disjoint union

$A = \{ (\sigma, \tau) \mid \text{dom } \sigma = \text{dom } \tau \}$

$(\sigma, \tau)$

$\sigma \rightarrow \tau$

$\sigma_1 \rightarrow \text{error}$ iff $\sigma_1 \upharpoonright \text{dom}(\sigma) = \sigma \upharpoonright \text{dom}(\sigma_1) \subset \sigma$

$(\sigma_1, \tau_1) \nmid (\sigma_2, \tau_2)$ iff $\text{dom}(\sigma_1) \cap \text{dom}(\sigma_2) = \emptyset$

cf. LICS 1996
EXAMPLE

action traces, shared store

\[ S = \text{Ide} \overset{\text{fin}}\longrightarrow V \]

- \( \bigotimes \) - disjoint union

\[ A = \{i=v, i:=v \mid i \in \text{Ide}, v \in V\} \]

\[ [i:v] \overset{i=v}\rightarrow [i:v] \]

\[ [i:v] \overset{i:=v'}\rightarrow [i:v'] \]

\[ \sigma \overset{i=v, i:=v'}\rightarrow \text{error} \quad \text{iff} \quad i \notin \text{dom}(\sigma) \]

\[ \neg(i:=v \neq i=v'), \neg(i:=v \neq i:=v') \]

cf. CONCUR 2002
EXAMPLE

action traces, shared mutable state

\[ S = \text{Store} \times \text{Heap} \]

- disjoint union, componentwise

\[ A = A_{\text{store}} \cup \{[l]=v, [l]:=v, \text{alloc}(l,v), \text{disp} l\} \]

\[
\begin{align*}
\text{alloc}(l,v) & : ([ ], [ ]) \rightarrow ([ ], [l:v]) \\
\text{disp} l & : ([ ], [l:v]) \rightarrow ([ ], [ ]) \\
\text{disp} l & : (s, h) \rightarrow \text{error} \text{ iff } l \notin \text{dom}(h) \\
\neg (\text{disp} l \not\equiv \text{disp} l)
\end{align*}
\]

cf. CONCUR 2004
Example

permissions

$S = \text{Ide}_\text{fin} V \times P$, $(P, \oplus, \top)$ a permission algebra

- $\otimes$ combines permissions, when compatible

$A = \{(i=v, \pi), (i:=v, \top) \mid \pi \in P, v \in V\}$

$[i:(v, \pi)]_{i=v, \pi} \rightarrow [i:(v, \pi)]$

$[i:(v, \top)]_{i:=v', \top} \rightarrow [i:(v', \top)]$

$\sigma_{i=v, \pi} \rightarrow \text{error} \iff i \notin \text{dom}(\sigma)$

$\sigma_{i:=v', \top} \rightarrow \text{error} \iff \neg \exists v. (i, (v, \top)) \in \sigma$

$(i=v, \pi_1) \not\# (i=v, \pi_2) \text{ when } \pi_1 \oplus \pi_2 \text{ defined}$

cf. MFPS’05
Traces

A trace is a finite or infinite sequence of actions

\( \alpha \) is (sequentially) executable iff \( \exists \sigma. \sigma \Rightarrow \alpha \).

Let \( \alpha \bowtie \beta \) iff \( \alpha \beta \) executable

Let \( \text{Tr}(A) \subseteq \mathcal{P}(A^\infty) \) be sets of executable traces
**Semantic Recipe**
for sequential programs

★ Given a state model \( \Sigma = (S, A, \rightarrow, \#) \),
we can define a *trace semantics*

\[
\begin{align*}
\llbracket - \rrbracket_\Sigma & : \text{Com} \rightarrow \text{Tr}(A) \\
\llbracket - \rrbracket_\Sigma & : \text{Exp}_{int} \rightarrow \text{Tr}(A) \times V_{int} \\
\llbracket - \rrbracket_\Sigma & : \text{Exp}_{bool} \rightarrow \text{Tr}(A) \times V_{bool}
\end{align*}
\]

by structural induction

★ \( \llbracket c \rrbracket_\Sigma \) is set of executable traces
SEMANTIC CLAUSES

\[ [c_1; c_2] = [c_1][c_2] = \{ \alpha \beta \mid \alpha \in [c_1], \beta \in [c_2], \alpha \bowtie \beta \} \]

\[ [\text{while } b \text{ do } c] = ([b]_{\text{true}}[c])^* [b]_{\text{false}} \cup ([b]_{\text{true}}[c])^\omega \]
fault-avoiding correctness

**Definition**

\{p\}c\{q\} is **valid** iff

\[ \forall \sigma \in S. \forall \alpha \in [c]. \forall \sigma'. \]

\[ \sigma \models p \ \& \ \sigma \xRightarrow{\alpha} \sigma' \implies \sigma' \neq error \ \& \ \sigma' \models q \]

*every finite execution of c, from a state satisfying p, is error-free, and ends in a state satisfying q*
**Validation Theorem**

For all sequential programs, 
\[
\begin{align*}
\llbracket c_1 \rrbracket &= \llbracket c_2 \rrbracket \\
&\text{implies} \\
\forall C. \ \forall p, q. \\
\{p\} C[c_1]\{q\} \text{ valid iff } \{p\} C[c_2]\{q\} \text{ valid}
\end{align*}
\]

*sequential commands with the same executable traces satisfy the same formulas, in all sequential contexts*
Parallel programs

- $c_1 \parallel c_2$
  - shared memory
- with $r$ when $b$ do $c$
  - conditional critical region
- resource $r$ in $c$
  - local resource

$r \in \text{Res} =$ set of resource names
Resource actions

- $\Delta = \{\text{try } r, \text{ acq } r, \text{ rel } r \mid r \in \text{Res}\}$

- Each resource is exclusive
  - acquired by at most one process at a time
  - available when not currently acquired
  - process must acquire before release, keeps trying when unavailable
A sequence $\alpha \in (A \cup \Delta)^\infty$ is well-resourced iff

$$\forall r. \alpha \upharpoonright \{\text{acq } r, \text{ rel } r\} \leq (\text{acq } r \text{ rel } r)^\omega$$

acquires before releases
Ability to do resource actions depends on resource sets $R_1$ held by process, $R_2$ held by environment.

These sets start empty and stay disjoint...

\[
\begin{align*}
\text{acq } r & \quad R_1 \Rightarrow R_1 \cup \{r\} \quad \text{iff} \quad r \notin R_1 \cup R_2 \\
\text{rel } r & \quad R_1 \Rightarrow R_1 - \{r\} \quad \text{iff} \quad r \in R_1 \\
\text{try } r & \quad R_1 \Rightarrow R_1
\end{align*}
\]
Concurrent execution of non-independent actions may yield unpredictable results.

Introduce an action `abort` to model such races.

Let \[ A^\dagger = \text{def} \ A \cup \{\text{abort}\} \]

Define \[ \sigma \rightarrow \sigma' \ \text{iff} \ \sigma' = \text{error} \]
Let $\text{Tr}(A, \Delta)$ be sets of well-resourced traces over $A^\dagger \cup \Delta$

$$\text{Tr}(A, \Delta) \subseteq \wp(A^\dagger \cup \Delta)$$

A parallel program will denote a set of well-resourced traces

$$\llbracket \cdot \rrbracket : \text{Com} \rightarrow \text{Tr}(A, \Delta)$$
Parallel composition

\[ [c_1 || c_2] = [c_1]_\emptyset || [c_2]_\emptyset \]

* Can be characterized as a greatest fixed point

* \( \alpha \stackrel{R_1}{\parallel} \beta \) resource-sensitive, race-detecting fair merges

\[
(\lambda_1 \alpha) \stackrel{R_1 R_2}{\parallel} (\lambda_2 \beta)
= \{ \lambda_1 \gamma \mid R_1 \xrightarrow{\lambda_1} R'_1, \gamma \in \alpha \stackrel{R'_1 R_2}{\parallel} (\lambda_2 \beta) \}
\]

\[
\cup \{ \lambda_2 \gamma \mid R_2 \xrightarrow{\lambda_2} R'_2, \gamma \in (\lambda_1 \alpha) \stackrel{R'_2 R_1}{\parallel} \beta \}
\]

\[ \cup \{ \text{abort} \mid \neg (\lambda_1 \neq \lambda_2) \} \]
\[\text{region} \]

\[
\left[ \text{with } r \text{ when } b \text{ do } c \right] = \text{wait}^* \text{enter} \cup \text{wait}^\omega
\]

- wait = \{\text{try } r\} \cup (\text{acq } r) \lbrack b \rbrack_{false} (\text{rel } r)

- enter = (\text{acq } r) \lbrack b \rbrack_{true} \lbrack c \rbrack (\text{rel } r)
LOCAL RESOURCE

\[ [\text{resource } r \text{ in } c] = \{ \alpha \setminus r \mid \alpha \in \llbracket c \rrbracket_r \} \]

- \( \alpha \setminus r \) obtained by erasing \{acq r, rel r, try r\}
- \( \alpha \in \llbracket c \rrbracket_r \) iff \( \alpha \in \llbracket c \rrbracket \) and

\[ \alpha \setminus r \leq (\text{acq } r \text{ (try } r)^\infty \text{ rel } r)^\infty \]

resource not accessible by environment
Fault-avoiding correctness

Definition

\{p\}c\{q\} is **valid** iff

\[\forall \sigma \in S. \forall \alpha \in [c]. \forall \sigma'.\]

\[\sigma \models p \land \sigma \vdash \alpha \sigma' \text{ implies } \sigma' \neq \text{error} \land \sigma' \models q\]

(as before)
For all parallel programs,

$$[c_1] = [c_2]$$

implies

$$\forall C. \forall p,q. \{p\} C[c_1] \{q\} \text{ valid} \iff \{p\} C[c_2] \{q\} \text{ valid}$$

\textit{parallel commands with the same traces satisfy the same formulas, in all parallel contexts}
## Examples

... rational reconstruction

### State Model

<table>
<thead>
<tr>
<th>Global $S \times S$</th>
<th>Global transition traces</th>
<th>Park ’79</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local $S \times S$</td>
<td>Local transition traces</td>
<td>LICS ’96</td>
</tr>
<tr>
<td>Reads/writes</td>
<td>Store action traces</td>
<td>CONCUR’ 02</td>
</tr>
<tr>
<td>Store+heap</td>
<td>Store/heap action traces</td>
<td>CONCUR’ 04</td>
</tr>
<tr>
<td>Permissive state</td>
<td>Permissive action traces</td>
<td>MFPS ’05</td>
</tr>
</tbody>
</table>
Executable traces

- Validity of \{p\}c\{q\} depends only on the *executable* traces of c

- But the *executable* traces of \(c_1 \parallel c_2\) cannot be derived from the *executable* traces of \(c_1\) and \(c_2\)

- So our semantic recipe for \(c_1 \parallel c_2\) includes *non-sequential* traces

- But how non-sequential do we need to be?
Dijkstra’s principle

- A rule for designing correct concurrent programs
  
  “... regard processes as independent, except when they synchronize”

- Suggests working with “almost sequential” traces...
Almost sequential
... sequential except at synchronizations

- A trace $\alpha$ is *almost sequential* iff
  
  $$\alpha\backslash\{\text{try, rel}\} = \alpha_1 (\text{acq } r_1) \alpha_2 (\text{acq } r_2) ...$$
  
  where each $\alpha_n \in A^\infty$ is *sequential*

- The *almost sequential* traces of $c_1 \| c_2$ are fair merges of *almost sequential* traces of $c_1$ and $c_2$

- Easy to adjust semantic clauses to obtain just the *almost sequential* traces

  $$\llbracket c \rrbracket_{a\circ} \subseteq \llbracket c \rrbracket$$
Validation Theorem (improved)

For all parallel programs,

\[ [c_1]_{a^*} = [c_2]_{a^*} \]

implies

\[ \forall C. \ \forall p, q. \quad \{p\}C[c_1]\{q\} \text{ valid iff } \{p\}C[c_2]\{q\} \text{ valid} \]

*parallel commands with the same almost sequential traces satisfy the same formulas, in all parallel contexts*
**Equivalent traces**

... same effect, same resource protocol, in all contexts

- For $\alpha, \beta \in A^{\infty}$ let $\alpha \approx \beta$ iff

  $|\alpha| = |\beta|$ and $\forall \lambda. (\alpha \not\equiv \lambda \Rightarrow \beta \not\equiv \lambda)$

  where $|\alpha| = \{(\sigma, \sigma') \mid \sigma \xrightarrow{\alpha} \sigma'\}$

- Extend to Tr$(A, \Delta)$ so that $\alpha \approx \beta$ iff

  $\alpha = \alpha_1 \delta_1 \ldots \alpha_n \delta_n \ldots$

  $\beta = \beta_1 \delta_1 \ldots \beta_n \delta_n \ldots$

  where each $\alpha_i \in (A^\dagger)^{\infty}$, $\delta_i \in \Delta^+$

  and $\forall n. \alpha_n \approx \beta_n$
EQUIVALENT TRACE SETS

Let $T_1 \approx T_2$ iff

\[ \forall \alpha \in T_1. \exists \beta \in T_2. \alpha \approx \beta \]

and

\[ \forall \beta \in T_2. \exists \alpha \in T_1. \alpha \approx \beta \]
Validation theorem
(improved again)

For all parallel programs,

$$[c_1] \approx [c_2]$$

implies

$$\forall C. \forall p, q. \{p\} C[c_1] \{q\} \text{ valid} \iff \{p\} C[c_2] \{q\} \text{ valid}$$

*parallel commands with equivalent trace sets satisfy the same formulas, in all parallel contexts*
Footstep traces

- Obtained from *action trace* model by quotient
- Traces have form
  \[(\sigma_1, \sigma_1') X_1 \delta_1 (\sigma_1, \sigma_1') X_2 \delta_2 \ldots\]
  where each \(X_i\) is a *read-only* set
- For all parallel programs
  \[[c_1]_{\sigma'} = [c_2]_{\sigma'} \iff [c_1]_{a_\sigma} \approx [c_2]_{a_\sigma}\]

*cf. MFPS ’06*
Advantages

- For a *synchronization-free* parallel program, the footstep traces form a non-deterministic relation on states.
- Taming the combinatorial explosion
Validation theorem
(final version)

For all parallel programs
\[
[c_1]_{f_0} = [c_2]_{f_0}
\]
implies
\[
\forall C. \forall p,q. \{p\}C[c_1]\{q\} \text{ valid iff } \{p\}C[c_2]\{q\} \text{ valid}
\]

parallel commands with the same footstep traces satisfy the same formulas, in all parallel contexts
Compositionality

- Semantic model is compositional and supports reasoning about fault-avoiding partial correctness

- But partial correctness properties of $c_1 \parallel c_2$ cannot be deduced from partial correctness properties of $c_1$ and $c_2$

- For a compositional logic, we need to work with more general formulas
  - fault-avoiding rely/guarantee properties
Fault-avoiding logics

\[ \Gamma \vdash \{p\}c\{q\} \]

- \( \Gamma \) specifies protection rules and resource invariants
- Rely/guarantee interpretation...

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

- Implies fault-avoiding correctness
Examples

- Separation logic
  - sequential pointer-programs
  - Reynolds

- Simple shared memory
  - shared memory parallel, no pointers
  - Owicki/Gries

- Concurrent separation logic
  - shared memory parallel, pointers
  - O’Hearn

- Permissions logic
  - shared memory parallel, pointers
  - Bornat et al
Validity

Definition

\( \Gamma \vdash \{p\}c\{q\} \) is **valid** iff

\[
\forall \sigma \in S_\Gamma. \forall \alpha \in \llbracket c \rrbracket. \forall \sigma'.
\]

\[
\sigma \models p \land \sigma \xrightarrow{\alpha} \Gamma \sigma' \text{ implies } \sigma' \neq \text{error} \land \sigma' \models q
\]

every finite interactive execution of \( c \), in an environment that respects \( \Gamma \), from a state satisfying \( p \), respects \( \Gamma \), is error-free, and ends in a state satisfying \( q \)
INTERACTIVE VALIDATION

THEOREM

For all parallel programs

$\llbracket c_1 \rrbracket_{f^*} = \llbracket c_2 \rrbracket_{f^*}$

implies

$\forall C. \forall \Gamma, p, q.$

$\Gamma \vdash \{p\} C[c_1]\{q\}$ valid iff $\Gamma \vdash \{p\} C[c_2]\{q\}$ valid

parallel commands with
the same footstep traces
satisfy the same rely/guarantee formulas,
in all parallel contexts
SEPARATION LOGIC

* S = Store × Heap

* (s,h) ⊨ p₁ ⋆ p₂ iff  ∃s₁, s₂. ∃h₁ ⊥ h₂. s = s₁ ∪ s₂, h = h₁ ⊔ h₂,
  (s₁,h₁) ⊨ p₁ \& (s₂,h₂) ⊨ p₂

* p is precise iff

  \forall (s,h). ∃ at most one h′ ⊲ h
  such that (s,h′) ⊨ p

* Γ = r₁(X₁):I₁, ..., rₙ(Xₙ):Iₙ
  Xᵢ disjoint, Iᵢ precise, ...
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

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

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

RESOURCE RULE

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

- Each rule of concurrent separation logic is valid

Use semantic model to formalize
- local state
- ownership transfer

Proof reveals key role of precision
Similarly...

- Soundness proofs for
  - Owicki-Gries permissions logic

  based on appropriate choice of state model
Conclusions

- A general, abstract notion of *state model*
- A recipe for constructing semantic models
- Suitable for compositional reasoning
  - fault-avoiding partial correctness
  - rely/guarantee partial correctness properties
- Soundness proofs for fault-avoiding logics
Future research

- Fault-avoiding logics
  - total correctness
  - safety and liveness
- Semantic models
  - full abstraction?
- Synchronization
  - other primitives
  - abstract model?