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*

State and Action

Definition

A ***state model*** is a tuple $(S, A, \rightarrow, \#)$

with

$S = (S, \otimes)$	*states*
A	*actions*
$\rightarrow \subseteq S \times A \times S^{\dagger}$	*footprint*
$S^{\dagger} = S \uplus \{error\}$	

and

$\otimes : S \times S \rightharpoonup S$	*compatibility*
$\# \subseteq A \times A$	*independence*

satisfying natural axioms ...

State axioms

- (S, ⊗) *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 σ, λ
at most one σ_1 such that
$$\sigma_1 \xrightarrow{\lambda} \sigma_1' \ , \ \sigma = \sigma_1 \otimes \sigma_2$$

- ***Failure is irrevocable***

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

Independence axioms

- ***Independence implies non-interfering footprints***

$$\lambda_1 \# \lambda_2 \ \ \& \ \sigma_1 \xrightarrow{\lambda_1} \sigma_1' \ \& \ \sigma_2 \xrightarrow{\lambda_2} \sigma_2'$$
$$\& \ \ \sigma = \sigma_1 \otimes \tau_1 = \sigma_2 \otimes \tau_2$$

implies

$$\exists \tau_1', \tau_2'. \ \ \sigma_1' \otimes \tau_1 = \sigma_2 \otimes \tau_2'$$
$$\& \ \ \sigma_2' \otimes \tau_2 = \sigma_1 \otimes \tau_1'$$
$$\& \ \ \sigma_1' \otimes \tau_1' = \sigma_2' \otimes \tau_2'$$

- ***Symmetry***

$\lambda_1 \# \lambda_2$ implies $\lambda_2 \# \lambda_1$

Enabling

For any state model we can derive an *enabling* relation

$$\Rightarrow \;\subseteq\; S^{\dagger} \times A \times S^{\dagger}$$

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

$$\sigma = \sigma_1 \otimes \sigma_2$$

$$\&\quad \sigma' = \sigma_1' \otimes \sigma_2$$

$$\&\quad \sigma_1 \overset{\lambda}{\rightarrow} \sigma_1'$$

Let $\sigma \overset{\lambda}{\Rightarrow} \mathit{error}$ iff $\sigma \overset{\lambda}{\rightarrow} \mathit{error}$ or $\sigma = \mathit{error}$

CONSEQUENCES

- Frame

$$\sigma_1 \overset{\lambda}{\Rightarrow} \tau_1 \neq \mathit{error} \quad \& \quad \sigma_1 \otimes \sigma_2 \overset{\lambda}{\Rightarrow} \tau$$

implies

$$\tau = \tau_1 \otimes \sigma_2$$

- Safety monotonicity

$$\sigma_1 \otimes \sigma_2 \overset{\lambda}{\Rightarrow} \mathit{error}$$

implies

$$\sigma_1 \overset{\lambda}{\Rightarrow} \mathit{error}$$

CONSEQUENCES

- ***Independent actions don't interfere***

If $\lambda_1 \# \lambda_2$ then

$$\sigma \overset{\lambda_1}{\Rightarrow} \tau_1 \,,\; \sigma \overset{\lambda_2}{\Rightarrow} \tau_2$$

implies

$$\exists \tau. \quad \tau_1 \overset{\lambda_2}{\Rightarrow} \tau \,,\; \tau_2 \overset{\lambda_1}{\Rightarrow} \tau$$

EXAMPLE

global transition traces

- $S = (\mathrm{Ide} \to V) \cup \{\mathbb{1}\}$
- $\sigma \otimes \mathbb{1} = \sigma = \mathbb{1} \otimes \sigma$
- $A = S \times S$
- $\sigma \xrightarrow{(\sigma, \tau)} \tau$
- $(\sigma_1, \tau_1) \sharp (\sigma_2, \tau_2)$ iff $\sigma_1 = \tau_1 = \sigma_2 = \tau_2$

cf. Park 1979

EXAMPLE

local transition traces

- $S = \mathrm{Ide} \rightharpoonup_{fin} V$
- $-\otimes-$ disjoint union
- $A = \{(\sigma, \tau) \mid \mathrm{dom}\ \sigma = \mathrm{dom}\ \tau\}$
- $\sigma \xrightarrow{(\sigma, \tau)} \tau$
- $\sigma_1 \xrightarrow{(\sigma, \tau)} error$ iff $\sigma_1 \restriction \mathrm{dom}(\sigma) = \sigma \restriction \mathrm{dom}(\sigma_1) \subset \sigma$
- $(\sigma_1, \tau_1) \ \# \ (\sigma_2, \tau_2)$ iff $\mathrm{dom}(\sigma_1) \cap \mathrm{dom}(\sigma_2) = \emptyset$

cf. LICS 1996

EXAMPLE

action traces, shared store

- $S = Ide \rightharpoonup_{fin} V$
- $-\otimes-$ disjoint union
- $A = \{i{=}v,\ i{:=}v \mid i \in Ide,\ v \in V\}$
- $[i{:}v] \xrightarrow{i=v} [i{:}v]$
- $[i{:}v] \xrightarrow{i:=v'} [i{:}v']$
- $\sigma \xrightarrow{i=v,\ i:=v'} \mathit{error}$ iff $i \notin dom(\sigma)$
- $\neg(i{:=}v \ \# \ i{=}v'),\ \neg(i{:=}v \ \# \ i{:=}v')$

cf. CONCUR 2002

EXAMPLE

action traces, shared mutable state

- S = Store × Heap
- -⊗- disjoint union, componentwise
- $A = A_{store} \cup \{[l]{=}v,\ [l]{:=}v,\ alloc(l,v),\ disp\ l\}$
- $([\ ],[\ ]) \xrightarrow{alloc(l,v)} ([\ ],[l{:}v])$
- $([\ ],[l{:}v]) \xrightarrow{disp\ l} ([\ ],[\ ])$
- $(s,h) \xrightarrow{disp\ l}$ *error* iff $l \notin dom(h)$
- $\neg(disp\ l \,\#\, disp\ l)$

cf. CONCUR 2004

Example

permissions

- $S = Ide \to_{fin} V \times \mathbb{P}$, $(\mathbb{P}, \oplus, \top)$ a *permission algebra*
- $-\otimes-$ *combines* permissions, when *compatible*
- $A = \{(i{=}v, \pi), (i{:=}v, \top) \mid \pi \in \mathbb{P}, v \in V\}$
- $[i{:}(v,\pi)] \xrightarrow{i=v,\pi} [i{:}(v,\pi)]$
- $[i{:}(v,\top)] \xrightarrow{i:=v',\top} [i{:}(v',\top)]$
- $\sigma \xrightarrow{i=v,\pi} \textit{error}$ iff $i \notin dom(\sigma)$
- $\sigma \xrightarrow{i:=v',\top} \textit{error}$ iff $\neg \exists v. (i, (v,\top)) \in \sigma$
- $(i{=}v,\pi_1) \sharp (i{=}v,\pi_2)$ when $\pi_1 \oplus \pi_2$ defined

cf. MFPS'05

Traces

- A trace is a finite or infinite sequence of actions
- α is (sequentially) *executable* iff $\exists\sigma.\ \sigma \overset{\alpha}{\Rightarrow} \cdot$
- Let $\alpha \frown \beta$ iff $\alpha\beta$ executable
- Let $\mathrm{Tr}(A) \subseteq \wp(A^\infty)$ be sets of executable traces

Semantic recipe

for sequential programs

- Given a state model $\Sigma = (S, A, \rightarrow, \sharp)$ we can define a *trace semantics*

$$[\![-]\!]_\Sigma : \quad Com \rightarrow Tr(A)$$

$$[\![-]\!]_\Sigma : \quad Exp_{int} \rightarrow Tr(A) \times V_{int}$$

$$[\![-]\!]_\Sigma : \quad Exp_{bool} \rightarrow Tr(A) \times V_{bool}$$

  by structural induction

- $[\![c]\!]_\Sigma$ is set of executable traces

SEMANTIC CLAUSES

- $[\![c_1;c_2]\!] = [\![c_1]\!]\,[\![c_2]\!]$

  $= \{\alpha\beta \mid \alpha\in[\![c_1]\!],\ \beta\in[\![c_2]\!],\ \alpha\frown\beta\}$

- $[\![\textbf{while}\ b\ \textbf{do}\ c]\!] =$

  $([\![b]\!]_{\textbf{true}}[\![c]\!])^{*}\ [\![b]\!]_{\textbf{false}} \cup ([\![b]\!]_{\textbf{true}}[\![c]\!])^{\omega}$

Fault-avoiding correctness

Definition

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

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

$$\sigma \models p\ \&\ \sigma \overset{\alpha}{\Rightarrow} \sigma' \text{ implies } \sigma' \neq \mathit{error}\ \&\ \sigma' \models q$$

*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,

$$[\![c_1]\!] = [\![c_2]\!]$$

implies

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

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

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

Parallel programs

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

$r \in Res$ = set of *resource names*

Resource actions

- Δ = {try r, acq r, rel r | r ∈ 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

WELL-RESOURCED

- 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

RESOURCE CONSTRAINTS

- 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...

$$R_1 \underset{R_2}{\overset{\text{acq } r}{\Rightarrow}} R_1 \cup \{r\} \quad \text{iff} \quad r \notin R_1 \cup R_2$$

$$R_1 \underset{R_2}{\overset{\text{rel } r}{\Rightarrow}} R_1 - \{r\} \quad \text{iff} \quad r \in R_1$$

$$R_1 \underset{R_2}{\overset{\text{try } r}{\Rightarrow}} R_1$$

RACE CONDITIONS

- Concurrent execution of non-independent actions may yield unpredictable results
- Introduce an action *abort* to model such races
- Let $A^{\dagger} =_{def} A \cup \{abort\}$
- Define $\sigma \xrightarrow{abort} \sigma'$ iff $\sigma' = error$

Semantic recipe
for parallel programs

- Let $\mathrm{Tr}(A, \Delta)$ be sets of well-resourced traces over $A^{\dagger} \cup \Delta$

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

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

$$[\![-]\!] : \mathrm{Com} \to \mathrm{Tr}(A, \Delta)$$

Parallel composition

$$[\![c_1 \| c_2]\!] = [\![c_1]\!]_{\emptyset}\|_{\emptyset}[\![c_2]\!]$$

- $\alpha \;{}_{R_1}\|_{R_2}\; \beta$ ***resource-sensitive, race-detecting*** fair merges
- Can be characterized as a *greatest fixed point*

$$(\lambda_1\alpha)\;{}_{R_1}\|_{R_2}\;(\lambda_2\beta) =$$

$$\{\lambda_1\gamma \mid R_1 \overset{\lambda_1}{\underset{R_2}{\Rightarrow}} R'_1,\ \gamma \in \alpha\;{}_{R'_1}\|_{R_2}\;(\lambda_2\beta)\}$$

$$\cup \quad \{\lambda_2\gamma \mid R_2 \overset{\lambda_2}{\underset{R_1}{\Rightarrow}} R'_2,\ \gamma \in (\lambda_1\alpha)\;{}_{R_1}\|_{R'_2}\;\beta\}$$

$$\cup \quad \{\, abort \mid \neg(\lambda_1 \# \lambda_2)\,\}$$

REGION

$$[\![\textbf{with } r \textbf{ when } b \textbf{ do } c]\!] = \text{wait}^* \text{ enter} \cup \text{wait}^\omega$$

- $\text{wait} = \{\text{try } r\} \cup (\text{acq } r)[\![b]\!]_{\text{false}}(\text{rel } r)$
- $\text{enter} = (\text{acq } r)\,[\![b]\!]_{\text{true}}\,[\![c]\!](\text{rel } r)$

LOCAL RESOURCE

$$[\![\textbf{resource}\ r\ \textbf{in}\ c]\!] = \{\ \alpha \setminus r \mid \alpha \in [\![c]\!]_r\ \}$$

- $\alpha \setminus r$ obtained by erasing {acq r, rel r, try r}
- $\alpha \in [\![c]\!]_r$ iff $\alpha \in [\![c]\!]$ and

  $$\alpha \upharpoonright r \leq (\text{acq } r\ (\text{try } r)^\infty\ \text{rel } r)^\infty$$

resource not accessible by environment

Fault-avoiding correctness

Definition

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

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

$$\sigma \models p \ \&\ \sigma \overset{\alpha}{\Rightarrow} \sigma' \ \text{implies}\ \sigma' \neq \mathit{error}\ \&\ \sigma' \models q$$

(as before)

Validation Theorem

For all parallel programs,

$$[\![c_1]\!] = [\![c_2]\!]$$

implies

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

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

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

EXAMPLES

... rational reconstruction

STATE MODEL → **SEMANTICS**

global S×S	global transition traces	*Park '79*
local S×S	local transition traces	*LICS '96*
reads/writes	store action traces	*CONCUR' 02*
store+heap	store/heap action traces	*CONCUR' 04*
permissive state	permissive action traces	*MFPS '05*

Executable traces

- Validity of {p}c{q} depends only on the ***executable*** traces of c
- But the *executable* traces of $c_1||c_2$ cannot be derived from the *executable* traces of c_1 and c_2
- So our semantic recipe for $c_1||c_2$ 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 α is ***almost sequential*** iff

  $$\alpha \backslash \{try, rel\} = \alpha_1\ (acq\ r_1)\ \alpha_2\ (acq\ r_2)\ \ldots$$

  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

  $$[\![c]\!]_{as} \subseteq [\![c]\!]$$

Validation Theorem

(improved)

For all parallel programs,

$$[\![c_1]\!]_{as} = [\![c_2]\!]_{as}$$

implies

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

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

  $$|\alpha| = |\beta| \text{ and } \forall\lambda.\ (\alpha \# \lambda \Leftrightarrow \beta \# \lambda)$$

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

- 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

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

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

$$[\![c_1]\!]_{fs} = [\![c_2]\!]_{fs} \quad \text{iff} \quad [\![c_1]\!]_{as} \approx [\![c_2]\!]_{as}$$

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

$$[\![c_1]\!]_{fs} = [\![c_2]\!]_{fs}$$

implies

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

$\{p\}C[c_1]\{q\}$ **valid** iff $\{p\}C[c_2]\{q\}$ **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 $c_1 || c_2$ 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

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

- Γ specifies *protection rules* and *resource invariants*
- Rely/guarantee interpretation...

> every finite ***interactive*** *execution* of c,
> in an ***environment that respects*** Γ,
> from a state satisfying p,
> ***respects*** Γ, is *error-free*,
> and ends in a state satisfying q

- Implies fault-avoiding correctness

Examples

- Separation logic — *Reynolds*
  sequential pointer-programs
- Simple shared memory — *Owicki/Gries*
  shared memory parallel, no pointers
- Concurrent separation logic — *O'Hearn*
  shared memory parallel, pointers
- Permissions logic — *Bornat et al*
  shared memory parallel, pointers

VALIDITY

Definition

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

$$\forall \sigma \in S_\Gamma .\ \forall \alpha \in [\![c]\!] .\ \forall \sigma'.$$

$$\sigma \models p \ \&\ \sigma \overset{\alpha}{\underset{\Gamma}{\Rightarrow}} \sigma' \ \text{ implies } \ \sigma' \neq \textit{error} \ \&\ \sigma' \models q$$

every finite ***interactive execution*** *of c,*
in an ***environment that respects*** *Γ,*
from a state satisfying p,
respects *Γ, is error-free,*
and ends in a state satisfying q

INTERACTIVE VALIDATION

THEOREM

For all parallel programs

$$[\![c_1]\!]_{fs} = [\![c_2]\!]_{fs}$$

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
- $(s,h) \vDash p_1 \star p_2$ iff $\exists s_1, s_2.\ \exists h_1 \perp h_2.$

  $s = s_1 \cup s_2,\quad h = h_1 \uplus h_2,$

  $(s_1,h_1) \vDash p_1 \ \&\ (s_2,h_2) \vDash p_2$

- p is *precise* iff

  $\forall (s,h).\ \exists$ at most one $h' \subseteq h$

  such that $(s,h') \vDash p$

- $\Gamma = r_1(X_1):I_1, \ldots, r_n(X_n):I_n$

  X_j disjoint, I_j precise, ...

PARALLEL RULE

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

provided

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

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

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

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

REGION RULE

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

RESOURCE RULE

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

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?