A DENOTATIONAL APPROACH TO RELEASE/ACQUIRE CONCURRENCY

**GOAL**
WHY RELEASE/ACQUIRE?

- RA is an important fragment of C11, enables decentralized architectures (POWER)
- First adaptation of Brookes’s traces to a relaxed-memory software model
- Intricate causal semantics, not overwhelmingly detailed

Threads can disagree about the order of writes (non-multi-copy-atomic)

- Supports flag-based synchronization (e.g. for signaling a data structure is ready)
- Supports important transformations (e.g. thread sequencing, write-read-reorder)
- Supports read-modify-write atomicity (e.g. atomic compare-and-swap)
WHY MONAD-BASED?

- **Standard**
- Program effects added modularly
- The core language remains exactly the same
- Higher-order programming built-in
- Rich toolkit of definitions, theorems, and techniques

### Structural transformations

\[
\text{if } K_{\text{pure}} \text{ then } M; P_1 \text{ else } M; P_2 \\
\cong M; \text{ if } K_{\text{pure}} \text{ then } P_1 \text{ else } P_2
\]

### Logical relations

“related inputs go to related outputs”

### Substitution lemma

syntax substitution ~ semantic context
e etc etc etc
DENOTATIONAL SEMANTICS

$\mathbf{[\cdot] : \text{Term} \to \text{Deno}}$

compose from subterms’ denotations

For example:

$\mathbf{[\text{let } x = M_1 \text{ in } M_2]} \triangleq \mathbf{[M_1 \rightsquigarrow \lambda x \cdot [M_2]}}$

$\mathbf{[M_1 \parallel M_2]} \triangleq \mathbf{[M_1 \parallel \parallel \parallel M_2]}$

Monadic bind

A modular effect extension
ADEQUACY

\[ \llbracket M \rrbracket \geq \llbracket K \rrbracket \implies M \rightarrow K \]

\( K \) denotationally refines \( M \)

\( K \) contextually refines \( M \)
safe to replace within any context

Abstraction:
We want this to hold as much as possible
ADEQUACY

With non-determinism as sets

\[
\text{Deno} = \mathcal{P}(\text{Behavior})
\]

\[\llbracket M \rrbracket \supseteq \llbracket K \rrbracket \implies M \rightarrow K\]

Every possible behavior of \( K \)

is a possible behavior of \( M \)

\( K \) contextually refines \( M \)
safe to replace within any context

\[\text{Abstraction: We want this to hold as much as possible}\]
GOAL

Design a standard, monad-based denotational semantics à la Moggi [1991]

Using Brookes-style [1996], totally-ordered traces

For weak, shared-memory model

RELEASE/ACQUIRE
TRACE-BASED SEMANTICS
Brookes [1996]

Main ingredient:
- linearly-ordered traces
- of local state-transitions
- that sequence and interleave

No interference  Possible interference

\[ \langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \cdots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle \]

\[ \langle \mu_1, \mu'_1 \rangle \langle \mu_2, \mu'_2 \rangle \cdots \langle \mu_n, \mu'_n \rangle \quad \langle \rho_1, \rho'_1 \rangle \langle \rho_2, \rho'_2 \rangle \cdots \langle \rho_n, \rho'_n \rangle \]
TRACE-BASED SEMANTICS

Brookes [1996]

Main ingredient:

- linearly-ordered traces
- of local state-transitions
- that sequence and interleave

No interference  Possible interference

\[
\langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \ldots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle
\]

\[
\langle \mu_1', \mu_1' \rangle \langle \mu_2', \mu_2' \rangle \ldots \langle \mu_n', \mu_n' \rangle \langle \rho_1, \rho_1' \rangle \langle \rho_2, \rho_2' \rangle \ldots \langle \rho_n, \rho_n' \rangle
\]
TRACE-BASED SEMANTICS

Brookes [1996]

Main ingredient:

- linearly-ordered traces
- of local state-transitions
- that sequence and interleave

No interference   Possible interference

\[ \langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \cdots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle \]

\[ \langle \rho_1, \rho'_1 \rangle \langle \mu_1, \mu'_1 \rangle \langle \mu_2, \mu'_2 \rangle \langle \rho_2, \rho'_2 \rangle \cdots \langle \mu_n, \mu'_n \rangle \langle \rho_n, \rho'_n \rangle \]
CONTRIBUTION

- **Standard** denotational semantics
- **Adequate** for Release/Acquire
- **Abstract** enough to verify every known RA-valid transformation in the literature (but no full-abstraction theorem)
- **Subtlety:** Rely/Guarantee interpretation of traces (our traces do not correspond directly to interrupted executions)
RELEASE/ACQUIRE
INTUITION VIA LITMUS TESTS

**Store Buffering**

\[
\begin{align*}
x & := 0; \\ y & := 0; \\ x & := 1; \\ y & := 1; \\ y? & \quad || \quad x?
\end{align*}
\]

**Message Passing**

\[
\begin{align*}
x & := 0; \\ y & := 0; \\ x & := 1; \\ y & := 1; \\ y? & \quad || \quad y?; \\ y & := 1 \quad || \quad x?
\end{align*}
\]
INTUITION VIA LITMUS TESTS

Store Buffering

\[
\begin{align*}
x &:= 0; y := 0; \\
x &:= 1; & y &:= 1; \\
y &\ ? / 0 & x &\ ? / 0
\end{align*}
\]

Message Passing

\[
\begin{align*}
x &:= 0; y := 0; \\
x &:= 1; & y &?; \\
y &:= 1 & x &?
\end{align*}
\]

\textit{Propagation is not instant}
INTUITION VIA LITMUS TESTS

Store Buffering

\[ x := 0; y := 0; \]
\[ x := 1; \]
\[ y := 1; \]
\[ y? //0 \]
\[ x? //0 \]

Message Passing

\[ x := 0; y := 0; \]
\[ x := 1; \]
\[ y := 1 \]
\[ y?; //1 \]
\[ x? //0 \]

Propagation is not instant

Propagation respects causality
**RELEASE/ACQUIRE VIEW-BASED OPERATIONAL SEMANTICS**

Kang et al. [2017]

- **Memory:** Timeline per location
  - Populated with immutable messages holding values
  - Each view points to msgs on each timeline
  - Threads have views — cannot read from “the past”
  - Msgs have views for enforcing causal propagation

---

![Diagram](image-url)

- Propagation is not instant
- Propagation respects causality
RELEASE/ACQUIRE VIEW-BASED OPERATIONAL SEMANTICS

Kang et al. [2017]

- **Memory**: Timeline per location
- Populated with immutable messages holding values
- Each view points to msgs on each timeline
- Threads have views — cannot read from “the past”
- Msgs have views for enforcing causal propagation

Propagation respects causality

Propagation is not instant
SUPPORTING FIRST-CLASS PARALLELISM

In the operational semantics

Traditional op-sem: static view-array

Laws of Parallel Programming, e.g. Left Neutrality

\[ \llbracket M \rrbracket = \llbracket (\langle \rangle \parallel M).snd \rrbracket \]

Write-Read Deorder (Crucial RA refinement)

\[ \llbracket x := 1; y? \rrbracket \supseteq \llbracket (x := 1 \parallel y?).snd \rrbracket \]

Extended op-sem: dynamic view-tree
TRACE-BASED SEMANTICS IN RA

\[ \alpha \langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \cdots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle \omega : r \]
TRACE-BASED SEMANTICS IN RA

\[ \alpha \langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \cdots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle \bowtie : r \]

Initial View
Sequence of Transitions
Final View
Returns

Before or ||

Rely On \( \mu_1 \) To Guarantee \( \rho_1 \)

Then

Rely On \( \mu_2 \) To Guarantee \( \rho_2 \)

Then...

Guarantee to the sequential environment to return \( r \)

After
TRACE-BASED SEMANTICS IN RA

\[ \alpha \langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \cdots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle \omega : \cdot r \]

Before

\[ \text{Rely on the sequential environment to reveal messages} \]

Guarantee to the sequential environment to reveal messages

Avoid including whole state in transitions

After

Initial View

Sequence of Transitions

Final View

Returns
RA DENOTATIONS

[ ] → [ ] : Term → Deno
MEMORY ACCESS

**Read**
\[ \alpha(x) \leq t \quad \vdash \quad v \in \mu \]
\[ \alpha \langle \mu, \mu \rangle \alpha \sqcup \beta \quad \vdash \quad v \in \begin{array}{l} x \, ? \end{array} \]

**Write**
\[ \alpha(x) < t \quad \rho = \mu \uplus \{ v : x \rightarrow t \} \]
\[ \alpha \langle \mu, \rho \rangle \alpha[x \rightarrow t] \quad \vdash \quad \langle \rangle \in \begin{array}{l} x := v \end{array} \]

**RMW**
Read the extended paper (°(^_^)°)

22
Sequential

\[ \alpha \xi_1 \kappa \cdot r_1 \in [[M_1]] \quad \kappa \xi_2 \omega \cdot r_2 \in [[M_2]] [x \mapsto r_1] \]

\[ \alpha \xi_1 \xi_2 \omega \cdot r_2 \in [[\text{let } x = M_1 \text{ in } M_2]] \]

SEQUENCING TRANSITIONS

Parallel

\[ \forall i \in \{1, 2\} \cdot \alpha \xi_i \omega \cdot r_i \in [[M_i]] \]

\[ \alpha \xi \omega \cdot \langle r_1, r_2 \rangle \in [[M_1 || M_2]] \]

INTERLEAVING TRANSITIONS
REWRITE CLOSURE RULES

Close denotations under rewrite rules

\(\pi \vdash r \in \| M \|\)

\(\pi \xrightarrow{x} \tau\)

\(\tau \vdash r \in \| M \|\)

Never introduced externally observable behavior

\(\alpha \langle \xi, \eta, \omega \rangle \xrightarrow{\text{stutter}} \alpha \langle \xi, \mu, \mu \rangle \eta \omega\)

\(\alpha \langle \xi, \mu, \rho \rangle \langle \rho, \theta \rangle \eta \omega \xrightarrow{\text{mumble}} \alpha \langle \xi, \mu, \theta \rangle \eta \omega\)

Propagate Reliance as a Guarantee

Rely on an omitted Guarantee

Brookes
REWRITE CLOSURE RULES

Close denotations under rewrite rules

\[ \pi \models r \in \| M \| \]
\[ \tau \models r \in \| M \| \]

Relying on more being revealed

\[ \alpha \leq \alpha' \]
\[ \alpha \xi \omega \xrightarrow{\text{rewind}} \alpha' \xi \omega \]

Guaranteeing less being revealed

\[ \omega \leq \omega' \]
\[ \alpha \xi \omega \xrightarrow{\text{forward}} \alpha \xi \omega' \]

Never introduced externally observable behavior
STRUCTURAL AND PARALLEL LAWS

Monad laws — structural equivalences for free, e.g. Hoisting

\[ [] \text{if } K_{\text{pure}} \text{ then } M; P_1 \text{ else } M; P_2 \] \[ = \] \[ [] M; \text{if } K_{\text{pure}} \text{ then } P_1 \text{ else } P_2 \]

Interleaving — properties of parallel composition, e.g. generalized sequencing

\[ [] (M_1; M_2) \parallel (K_1; K_2) \] \[ \supseteq \] \[ [] (M_1 \parallel K_1); (M_2 \parallel K_2) \]
Some transformations are valid due to more complicated reasons, e.g.:

**Redundant Read Elimination**

\[ y?; \ M \rightarrow \ M \]

holds due to delicate semantic invariants

**Overwritten Write Elimination**

\[ x := 0; x := 1 \rightarrow x := 1 \]

holds even though state diverges
DELICATE SEMANTIC INVARIANTS

Redundant Read Elimination

\[ y?; M \rightarrow M \]

we identify operational invariants
and impose them as denotational requirements

\[ \kappa \langle \mu, \mu \rangle \kappa \vdash \langle \rangle \in \llangle \langle \rangle \rrangle \quad \Rightarrow \quad \exists v. \kappa \langle \mu, \mu \rangle \kappa \vdash v \in \llangle y? \rrangle \]
OVERWRITTEN WRITE ELIMINATION

\[ x := 0; x := 1 \rightarrow x := 1 \]

\[ \alpha \langle \mu, \mu \uplus \{1\} \rangle \omega : \langle \rangle \]

\[ \Psi \]

\[ [[x := 0; x := 1]] \supseteq [[x := 1]] \]
DIVERGING STATE

Overwritten Write Elimination

\[ x := 0; x := 1 \rightarrow x := 1 \]

\[ \alpha \langle \mu, \mu \uplus \{ 1 \} \rangle \omega \vdash \langle \rangle \]

\[ [[x := 0; x := 1]] \supseteq [[x := 1]] \]

\[ \alpha \langle \mu, \mu \uplus \{ 0 \} \rangle \langle \mu \uplus \{ 0 \}, \mu \uplus \{ 01 \} \rangle \omega \vdash \langle \rangle \]
DIVerging STATE

Overwritten Write Elimination

\[ x := 0; x := 1 \rightarrow x := 1 \]

\[ \alpha \langle \mu, \mu \uplus \{ 1 \} \rangle \omega : \langle \rangle \]

\[ \uplus \]

\[ \lbrack x := 0; x := 1 \rbrack \supseteq \lbrack x := 1 \rbrack \]

\[ \uplus \]

\[ \alpha \langle \mu, \mu \uplus \{ 0 \} \rangle \langle \mu \uplus \{ 0 \}, \mu \uplus \{ 0, 1 \} \rangle \omega : \langle \rangle \]

\[ mumble \]
DIVERGING STATE

Overwritten Write Elimination

\[ x := 0; x := 1 \rightarrow x := 1 \]

\[ \alpha \langle \mu, \mu \uplus \{ 1 \} \rangle \omega : \langle \rangle \]

\[ \Psi \]

\[ \text{absorb} \]

\[ \alpha \langle \mu, \mu \uplus \{ 0 \} \rangle \langle \mu \uplus \{ 0 \}, \mu \uplus \{ 0, 1 \} \rangle \omega : \langle \rangle \]

\[ \text{mumble} \]

\[ [[x := 0; x := 1]] \supseteq [[x := 1]] \]
NO CORRESPONDENCE WITH INTERRUPTED EXECUTIONS

\[\alpha \langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \cdots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle \omega \vdash r \]

\[\langle \mu_2, - \rangle, M_1 \rightarrow^* \langle \rho_2, - \rangle, M_2 \cdots\]

Absorb
### ALL REWRITE RULES

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
<th>Symbol</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loosen</td>
<td>$\alpha \xi (\eta \uplus {\epsilon}) \omega$</td>
<td>$\rightarrow_{Ls}$</td>
<td>$\alpha \xi (\eta \uplus {\nu}) \omega$ $\nu \leq_{vw} \epsilon$</td>
</tr>
<tr>
<td>Expel</td>
<td>$\alpha \xi (\eta \uplus {\nu^{v.i}}) \omega$</td>
<td>$\rightarrow_{Ex}$</td>
<td>$\alpha \xi (\eta \uplus {\nu, \epsilon}) \omega$ $\nu \prec \epsilon$</td>
</tr>
<tr>
<td>Condense</td>
<td>$\alpha \xi (\eta \uplus {\nu, \epsilon}) \omega$</td>
<td>$\rightarrow_{Cn}$</td>
<td>$(\alpha \xi (\eta \uplus {\nu}) \omega) \uparrow \epsilon$ $\nu \subseteq \epsilon$</td>
</tr>
<tr>
<td>Stutter</td>
<td>$\alpha \xi \eta \omega$</td>
<td>$\rightarrow_{St}$</td>
<td>$\alpha \xi \langle \mu, \mu \rangle \eta \omega$</td>
</tr>
<tr>
<td>Mumble</td>
<td>$\alpha \xi \langle \mu, \rho \rangle \langle \rho, \theta \rangle \eta \omega$</td>
<td>$\rightarrow_{Mu}$</td>
<td>$\alpha \xi \langle \mu, \theta \rangle \eta \omega$</td>
</tr>
<tr>
<td>Tighten</td>
<td>$\alpha \xi \langle \mu, \rho \uplus {\nu} \rangle \eta \uplus {\nu} \omega$</td>
<td>$\rightarrow_{Ti}$</td>
<td>$\alpha \xi \langle \mu, \rho \uplus {\epsilon} \rangle \eta \uplus {\epsilon} \omega$ $\nu \leq_{vw} \epsilon$</td>
</tr>
<tr>
<td>Absorb</td>
<td>$\alpha \xi \langle \mu, \rho \uplus {\nu, \epsilon} \rangle \eta \uplus {\nu, \epsilon} \omega$</td>
<td>$\rightarrow_{Ab}$</td>
<td>$\alpha \xi \langle \mu, \rho \uplus {\nu^{v.i}} \rangle \eta \uplus {\nu^{v.i}} \omega$ $\nu \prec \epsilon$</td>
</tr>
<tr>
<td>Dilute</td>
<td>$(\alpha \xi \langle \mu, \rho \uplus {\nu} \rangle \eta \uplus {\nu} \omega) \uparrow \epsilon$</td>
<td>$\rightarrow_{Di}$</td>
<td>$\alpha \xi \langle \mu, \rho \uplus {\nu, \epsilon} \rangle \eta \uplus {\nu, \epsilon} \omega$ $\nu \subseteq \epsilon$</td>
</tr>
</tbody>
</table>

**Symbols and Notations**

- $\alpha$: Function symbol
- $\xi$, $\eta$, $\mu$, $\rho$, $\nu$, $\epsilon$: Variables
- $\uplus$: Union operator
- $\langle \cdot \rangle$: Indexed tuple
- $\forall_i$: For all $i$
- $\uparrow \epsilon$: Absorbing rewrite
- $\leq_{vw}$: Weak preordering
- $\prec$: Strict preordering
- $\subseteq$: Subset relation
NEW ADEQUACY PROOF IDEA

Traces are not operational — adequacy proof is significantly more challenging:

1. We first define a denotational semantics $\llbracket M \rrbracket$ but without the abstract rules

2. We show it is adequate — easier: traces correspond to interrupted executions (with an admissible view-advancing rule)

3. We show it is enough to apply the abstract closure $\dagger$ on top $\llbracket M \rrbracket = \llbracket M \rrbracket^\dagger$
   - This is the main technical challenge — complicated commutativity property

4. We show that the abstract rewrites preserve observable results (rather than interrupted executions)
Laws of Parallel Programming

Symmetry \[ M \parallel N \rightarrow \text{match } N \parallel M \text{ with } (y, x). \ (x, y) \]

Generalized Sequencing
\[(\text{let } x = M_1 \text{ in } M_2) \parallel (\text{let } y = N_1 \text{ in } N_2) \rightarrow \text{match } M_1 \parallel N_1 \text{ with } (x, y). \ M_2 \parallel N_2\]

Eliminations

Irrelevant Read \[ \ell? ; \langle \rangle \rightarrow \langle \rangle \]

Write-Write \[ \ell := v ; \ell := w \xrightarrow{\text{Ab}} \ell := w \]

Write-Read \[ \ell := v ; \ell? \rightarrow \ell := v ; v \]

Write-FAA \[ \ell := v ; \text{FAA } (\ell, w) \xrightarrow{\text{Ab}} \ell := (v + w) ; v \]

Read-Write \[ \text{let } x = \ell? \text{ in } \ell := (x + v) ; x \rightarrow \text{FAA } (\ell, v) \]

Read-Read \[ \langle \ell?, \ell? \rangle \rightarrow \text{let } x = \ell? \text{ in } \langle x, x \rangle \]

Read-FAA \[ \langle \ell?, \text{FAA } (\ell, v) \rangle \rightarrow \text{let } x = \text{FAA } (\ell, v) \text{ in } \langle x, x \rangle \]

FAA-Read \[ \langle \text{FAA } (\ell, v), \ell? \rangle \rightarrow \text{let } x = \text{FAA } (\ell, v) \text{ in } \langle x, x + v \rangle \]

FAA-FAA \[ \langle \text{FAA } (\ell, v), \text{FAA } (\ell, w) \rangle \xrightarrow{\text{Ab}} \text{let } x = \text{FAA } (\ell, v + w) \text{ in } \langle x, x + v \rangle \]

Others

Irrelevant Read Introduction \[ \langle \rangle \rightarrow \ell? ; \langle \rangle \]

Read to FAA \[ \ell? \xrightarrow{\text{Df}} \text{FAA } (\ell, 0) \]

Write-Read Deorder \[ \langle (\ell := v), \ell'? \rangle \xrightarrow{T_i} (\ell := v) \parallel \ell'? \] \[ (\ell \neq \ell') \]

Write-Read Reorder \[ \langle (\ell := v), \ell'? \rangle \xrightarrow{T_i} \text{let } x = \ell'? \text{ in } (\ell := v) ; x \] \[ (\ell \neq \ell') \]
CONCLUSION

- Standard, adequate and fully-compositional denotational semantic for RA
- Sufficiently abstract: validates all RA transformations that we know of (memory access, laws of parallel programming, structural transformations)
- More nuanced, complicated traces
  - interpreted as Rely/Guarantee sequences
  - denotations closed under 10 rewrite rules
- Extended RA view-based machine with compositional (i.e. first-class) parallelism (weak-memory models are usually studied with top-level parallelism)

OPPORTUNITIES

- Language features (e.g. recursion)
- Type-and-effect system (e.g. regions)
- Algebraic presentation (refines monad approach)
- Full C11 model (e.g. non-atomics)
- Full abstraction theorem (for first-order)?