Relaxed separation logic

Viktor Vafeiadis   Chinmay Narayan

MPI-SWS   IIT Delhi

Goal: Understand concurrent programs.

Tool: Concurrent program logics:

- **C**oncurrent **S**eparation **L**ogic
- OG, RG, RGSep, LRG, DG, CAP, CaReSL...

$* * *$ What about *weak memory models*? $* * *$

All *sane* memory models satisfy the DRF property:

Theorem (DRF-property)

If $[\![Prg]\!]_{\mathrm{SC}}$ *contains no data races, then* $[\![Prg]\!]_{\mathrm{Relaxed}} = [\![Prg]\!]_{\mathrm{SC}}$.

- Program logics that disallow data races are trivially sound.
- What about *racy* programs?

C11 operations

Two types of locations: ordinary and atomic

- Races on ordinary accesses $\rightsquigarrow$ undefined

Several kinds of atomic accesses:

- Sequentially consistent (reads & writes)
- Release (writes)
- Acquire (reads)
- Relaxed (reads & writes)

A few more advanced constructs:

- Fences, consume reads, ... (ignored here)

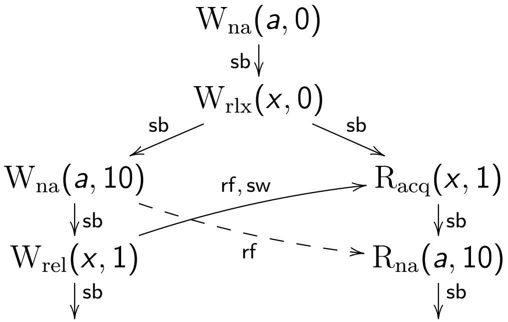
Execution = set of events & a few relations:

- sb: sequenced before
- rf: reads-from map
- mo: memory order per location
- sc: seq.consistency order
- sw: synchronizes with (derived)
  W-release $\xrightarrow{\text{rf}}$ R-acq $\implies$ W-release $\xrightarrow{\text{sw}}$ R-acq
- hb: happens before (derived, hb $\stackrel{\text{def}}{=} (\text{sb} \cup \text{sw})^{+}$)

Axioms constraining the *consistent* executions.

Message passing example

$$
\begin{array}{c}
[a]_{\mathrm{na}} := 0; \\
[x]_{\mathrm{rlx}} := 0; \\
\left(\begin{array}{l} [a]_{\mathrm{na}} := 10; \\ [x]_{\mathrm{rel}} := 1; \end{array} \;\middle\|\; \begin{array}{l} \textbf{if } ([x]_{\mathrm{acq}} = 1) \\ \quad \textbf{print } [a]_{\mathrm{na}}; \end{array}\right)
\end{array}
$$



Separation logic recap

$$[\![\ell \mapsto v]\!] \stackrel{\text{def}}{=} \{h \mid h(\ell) = v\}$$

$$[\![P_1 * P_2]\!] \stackrel{\text{def}}{=} \{h_1 \uplus h_2 \mid h_1 \in [\![P_1]\!] \wedge h_2 \in [\![P_2]\!]\}$$

Proof rules:

$$\{\ell \mapsto -\}\ [\ell] := v\ \{\ell \mapsto v\} \qquad \text{(WRI)}$$

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}} \qquad \text{(FRM)}$$

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \| C_2\ \{Q_1 * Q_2\}} \qquad \text{(PAR)}$$

Read-acquire & write-release permissions (1/2)

- Introduce two assertion forms:

$$P := \ldots \mid \ell \overset{\text{rel}}{\hookrightarrow} \mathcal{Q} \mid \ell \overset{\text{acq}}{\hookrightarrow} \mathcal{Q}$$

where $\mathcal{Q} \in \mathsf{Val} \to \mathsf{Assn}$.

- Initially (simplified rule):

$$\frac{\mathcal{Q}(v) = \mathsf{emp}}{\{\mathsf{emp}\}\ x := \mathbf{alloc}_{\text{atom}}(v)\ \{x \overset{\text{rel}}{\hookrightarrow} \mathcal{Q} * x \overset{\text{acq}}{\hookrightarrow} \mathcal{Q}\}}$$

- Release writes:

$$\{\mathcal{Q}(v) * \ell \overset{\text{rel}}{\hookrightarrow} \mathcal{Q}\}\ [\ell]_{\text{rel}} := v\ \{\ell \overset{\text{rel}}{\hookrightarrow} \mathcal{Q}\}$$

- Acquire reads:

$$\{\ell \overset{\text{acq}}{\hookrightarrow} \mathcal{Q}\}\ x := [\ell]_{\text{acq}}\ \{\mathcal{Q}(x) * \ell \overset{\text{acq}}{\hookrightarrow} \mathcal{Q}[x{:=}\mathsf{emp}]\}$$

where $\mathcal{Q}[x{:=}P] \overset{\text{def}}{=} \lambda y.\ \text{if } x{=}y \text{ then } P \text{ else } \mathcal{Q}(y)$.

- Splitting permissions:

$$\ell \overset{\text{rel}}{\hookrightarrow} \mathcal{Q} * \ell \overset{\text{rel}}{\hookrightarrow} \mathcal{Q} \iff \ell \overset{\text{rel}}{\hookrightarrow} \mathcal{Q}$$

$$\ell \overset{\text{acq}}{\hookrightarrow} \mathcal{Q}_1 * \ell \overset{\text{acq}}{\hookrightarrow} \mathcal{Q}_2 \iff \ell \overset{\text{acq}}{\hookrightarrow} (\mathcal{Q}_1 * \mathcal{Q}_2)$$

Simple ownership transfer example

Let $\mathcal{Q} := \{(0, \mathsf{emp}), (1, a \hookrightarrow 2)\}$.

$$\{\ \mathsf{emp}\ \}$$

$$a := \mathbf{alloc}_{\mathrm{na}}(0);\ \ x := \mathbf{alloc}_{\mathrm{atom}}(0);$$

$$\left\{\ a \hookrightarrow 0 * x \overset{\mathrm{rel}}{\hookrightarrow} \mathcal{Q} * x \overset{\mathrm{acq}}{\hookrightarrow} \mathcal{Q}\ \right\}$$

$\left\{\ a \hookrightarrow 0 * x \overset{\mathrm{rel}}{\hookrightarrow} \mathcal{Q}\ \right\}$	$\left\{\ x \overset{\mathrm{acq}}{\hookrightarrow} \mathcal{Q}\ \right\}$
$[a]_{\mathrm{na}} := 2;$	**repeat**
$\left\{\ a \hookrightarrow 2 * x \overset{\mathrm{rel}}{\hookrightarrow} \mathcal{Q}\ \right\}$	$\quad r := [x]_{\mathrm{acq}}$
$[x]_{\mathrm{rel}} := 1;$	$\quad \left\{\begin{array}{l} r = 0 * x \overset{\mathrm{acq}}{\hookrightarrow} \mathcal{Q} \vee \\ r = 1 * a \hookrightarrow 2 \end{array}\right\}$
$\{\ \mathsf{true}\ \}$	**until**$(r \neq 0);$
	$\{\ r = 1 * a \hookrightarrow 2\ \}$
	$r := [a]_{\mathrm{na}}$
	$\{\ r = 2 * a \hookrightarrow 2\ \}$

$$\{\ r = 2 * a \hookrightarrow 2\ \}$$

Basically, disallow ownership transfer.

- Relaxed reads:

$$\{\ell \overset{\mathrm{acq}}{\hookrightarrow} \mathcal{Q}\}\ x := [\ell]_{\mathrm{rlx}}\ \{\ell \overset{\mathrm{acq}}{\hookrightarrow} \mathcal{Q} \land (\mathcal{Q}(x) \neq \mathsf{false})\}$$

- Relaxed writes:

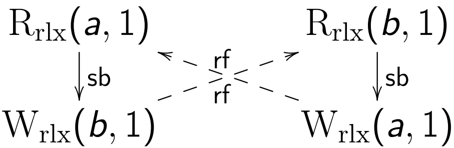
$$\frac{\mathcal{Q}(v) = \mathsf{emp}}{\{\ell \overset{\mathrm{rel}}{\hookrightarrow} \mathcal{Q}\}\ [\ell]_{\mathrm{rlx}} := v\ \{\ell \overset{\mathrm{rel}}{\hookrightarrow} \mathcal{Q}\}}$$

- Unsound in C11 because of dependency cycles.

$$
\begin{array}{c}
\textbf{let } a = \textbf{alloc}_{\mathrm{atom}}(0) \textbf{ in} \\
\textbf{let } b = \textbf{alloc}_{\mathrm{atom}}(0) \textbf{ in} \\
\begin{pmatrix} \textbf{if } 1 = [a]_{\mathrm{rlx}} \textbf{ then} \\ [b]_{\mathrm{rlx}} := 1 \end{pmatrix} \Bigg\| \begin{pmatrix} \textbf{if } 1 = [b]_{\mathrm{rlx}} \textbf{ then} \\ [a]_{\mathrm{rlx}} := 1 \end{pmatrix}
\end{array}
$$

A problematic consistent execution:

[*Initialization actions not shown*]



[Crude fix: Require $hb \cup rf$ to be acyclic.**]**

- New assertion form, $P := \ldots \mid \ell \overset{\mathrm{macq}}{\hookrightarrow} \mathcal{Q}$.
- Duplicable, $\ell \overset{\mathrm{macq}}{\hookrightarrow} \mathcal{Q} \iff \ell \overset{\mathrm{macq}}{\hookrightarrow} \mathcal{Q} * \ell \overset{\mathrm{macq}}{\hookrightarrow} \mathcal{Q}$.
- Proof rule for CAS:

$$\frac{\begin{array}{c} P \Rightarrow \ell \overset{\mathrm{macq}}{\hookrightarrow} \mathcal{Q} * \mathsf{true} \\ P * \mathcal{Q}(v) \Rightarrow \ell \overset{\mathrm{rel}}{\hookrightarrow} \mathcal{Q}' * \mathcal{Q}'(v') * R[v/z] \\ X \in \{\mathrm{rel}, \mathrm{rlx}\} \Rightarrow \mathcal{Q}(v) = \mathsf{emp} \\ X \in \{\mathrm{acq}, \mathrm{rlx}\} \Rightarrow \mathcal{Q}'(v') = \mathsf{emp} \\ \{P\}\ z := [\ell]_Y\ \{z \neq v \Rightarrow R\} \end{array}}{\{P\}\ z := CAS_{X,Y}(\ell, v, v')\ \{R\}}$$

Let $\mathcal{Q}_J(v) \stackrel{\text{def}}{=} (v = 0 \wedge \mathsf{emp}) \vee (v = 1 \wedge J)$

$Lock(x, J) \stackrel{\text{def}}{=} x \stackrel{\text{rel}}{\hookrightarrow} \mathcal{Q}_J * x \stackrel{\text{macq}}{\hookrightarrow} \mathcal{Q}_J$

$new\text{-}lock() \stackrel{\text{def}}{=}$

$\{\ J\ \}$

$res := \mathbf{alloc}_{\text{atom}}(1)$

$\{\ Lock(res, J)\ \}$

$unlock(x) \stackrel{\text{def}}{=}$

$\{\ J * Lock(x, J)\ \}$

$[x]_{\text{rel}} := 1$

$\{\ Lock(x, J)\ \}$

$lock(x) \stackrel{\text{def}}{=}$

$\{\ Lock(x, J)\ \}$

repeat

$\{\ Lock(x, J)\ \}$

$y := \mathsf{CAS}_{\text{acq,rlx}}(x, 1, 0)$

$$\left\{\ Lock(x, J) * \begin{pmatrix} y = 0 \wedge \mathsf{emp} \\ \vee\ y = 1 \wedge J \end{pmatrix}\ \right\}$$

until $y \neq 0$

$\{\ J * Lock(x, J)\ \}$

Technical challenges

- Assertions in heaps

  $\Longrightarrow$ Store syntactic assertions (modulo $*$-ACI)
- No (global) notions of state and time

  $\Longrightarrow$ Define a *logical* local notion of state

  $\Longrightarrow$ Annotate hb edges with logical state
- No operational semantics

  $\Longrightarrow$ Use the axiomatic semantics

  $\Longrightarrow$ Induct over max hb-path distance from top

Possible extensions / future work

- Take more advanced program logics
  (rely-guarantee, RGSep, deny-guarantee, . . .)
  and adapt them to C11 concurrency
- Handle the more advanced C11 constructs:
  consume atomics & fences
- Build a tool & verify real programs