

The Leaky Semicolon

Compositional Semantic Dependencies for Relaxed-Memory Concurrency

Alan Jeffrey* James Riely† Mark Batty‡
Simon Cooksey† Ilya Kaysin** Anton Podkopaev††

* Roblox † DePaul ‡ Kent

** JetBrains + Cambridge †† HSE

POPL
January 2022

The Leaky Semicolon

Compositional Semantic Dependencies for Relaxed-Memory Concurrency

Alan Jeffrey* James Riely† Mark Batty‡
Simon Cooksey† Ilya Kaysin** Anton Podkopaev††

* Roblox † DePaul ‡ Kent

** JetBrains + Cambridge †† HSE

POPL
January 2022

Remembering nothing that happened after 1999 ...

Remembering nothing that happened after 1999 ...

Develop a model of imperative programming

Remembering nothing that happened after 1999 ...

Develop a model of imperative programming that supports:

Shared-memory concurrency

Remembering nothing that happened after 1999 ...

Develop a model of imperative programming that supports:

- Shared-memory concurrency

- Compositional reasoning for sequencing and parallelism

Remembering nothing that happened after 1999 ...

Develop a model of imperative programming that supports:

- Shared-memory concurrency

- Compositional reasoning for sequencing and parallelism

- A few other things:

- Features: pointers, release/acquire/SC access, fences, RMWs

- Optimizations: reorderings, if-introduction, redundant read elimination, etc.

- Java Causality Test Cases

- DRF-SC, No-Thin-Air, temporal invariant reasoning

- Efficient implementation on Arm8

- Connection to C11, JavaScript, PTX, etc.

- Automatic litmus test checker

- Certified proofs

Sequential Computation

```
x := 0; r := x; y := r + 1
```

Sequential Computation

```
x := 0; r := x; y := r + 1
```

Preconditions!

Sequential Computation: Preconditions + Hoare Logic!

$x := 0; r := x; y := r + 1$

$y := r + 1 \quad \{y = 1\}$

[Hoare 1969]

Sequential Computation: Preconditions + Hoare Logic!

$x := 0; r := x; y := r + 1$

$\{r = 0\} \quad y := r + 1 \quad \{y = 1\}$

[Hoare 1969]

Sequential Computation: Preconditions + Hoare Logic!

$x := 0; r := x; y := r + 1$

$r := x \quad \{r = 0\}$
 $\{r = 0\} \quad y := r + 1 \quad \{y = 1\}$

[Hoare 1969]

Sequential Computation: Preconditions + Hoare Logic!

$x := 0; r := x; y := r + 1$

$\{x = 0\} \quad r := x \quad \{r = 0\}$
 $\{r = 0\} \quad y := r + 1 \quad \{y = 1\}$

[Hoare 1969]

Sequential Computation: Preconditions + Hoare Logic!

$x := 0; r := x; y := r + 1$

	$x := 0$	$\{x = 0\}$
$\{x = 0\}$	$r := x$	$\{r = 0\}$
$\{r = 0\}$	$y := r + 1$	$\{y = 1\}$

[Hoare 1969]

Sequential Computation: Preconditions + Hoare Logic!

$x := 0; r := x; y := r + 1$

$\{tt\}$	$x := 0$	$\{x = 0\}$
$\{x = 0\}$	$r := x$	$\{r = 0\}$
$\{r = 0\}$	$y := r + 1$	$\{y = 1\}$

[Hoare 1969]

Sequential Computation: Preconditions + Predicate Transformers!

$x := 0; r := x; y := r + 1$

$\text{tt} \Rightarrow wp(x := 0) (x = 0)$

$(x = 0) \Rightarrow wp(r := x) (r = 0)$

$(r = 0) \Rightarrow wp(y := r + 1) (y = 1)$

$(x = 0 \wedge r = 0) \rightarrow (y := r + 1, r := x)$

$(x = 0 \wedge r = 0) \rightarrow (y := r + 1, r := x)$

Concurrency?

Temporal Computation: Preconditions + Predicate Transformers!

Given:
Initial state: $x := 0; r := x; y := r$

Ordered Events!
 $(r = y) \rightarrow (y := r + 1)$

Concurrent Computation

```
x:=0; r:=x; y:=r+1 || x:=5
```

Concurrent Computation: Pomsets! (aka labeled partial orders)

$x := 0; r := x; y := r + 1 \parallel x := 5$

Wx0

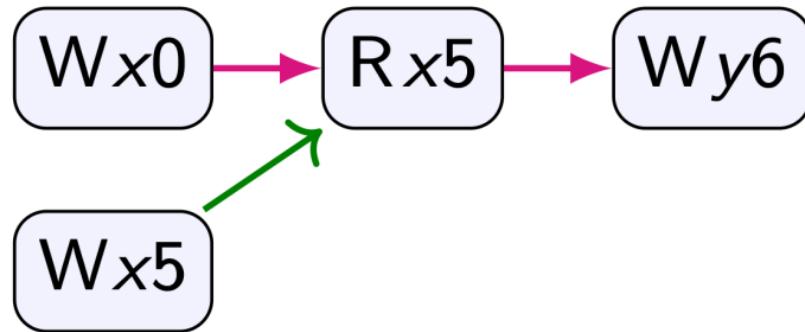
Rx5

Wy6

Wx5

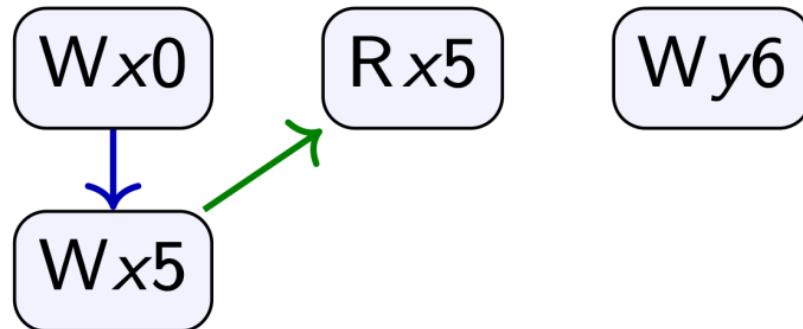
Concurrent Computation: Pomsets! (aka labeled partial orders)

$x := 0; r := x; y := r + 1 \parallel x := 5$



Concurrent Computation: Pomsets! (aka labeled partial orders)

$x := 0; r := x; y := r + 1 \parallel x := 5$



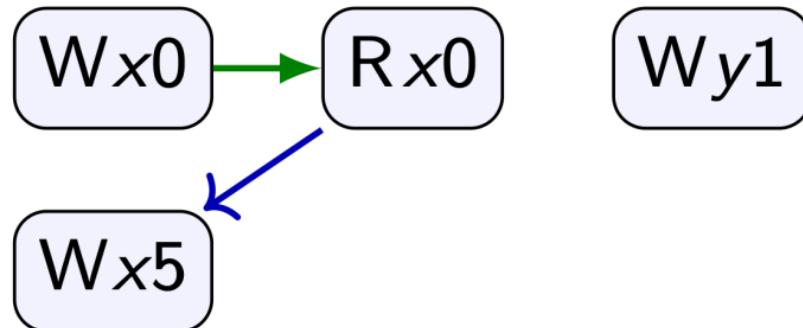
Complete pomset: $\forall R xv$

$\exists W xv < Rxv$

$\forall W xu$ either $W xu \leq W xv$ or $R xv < W xu$

Concurrent Computation: Pomsets! (aka labeled partial orders)

$x := 0; r := x; y := r + 1 \parallel x := 5$



Complete pomset: $\forall R xv$

$\exists W xv < Rxv$

$\forall W xu$ either $W xu \leq W xv$ or $R xv < W xu$

Current Computation: Pomsets! (aka labeled partial orders)

Hardware Reordering?

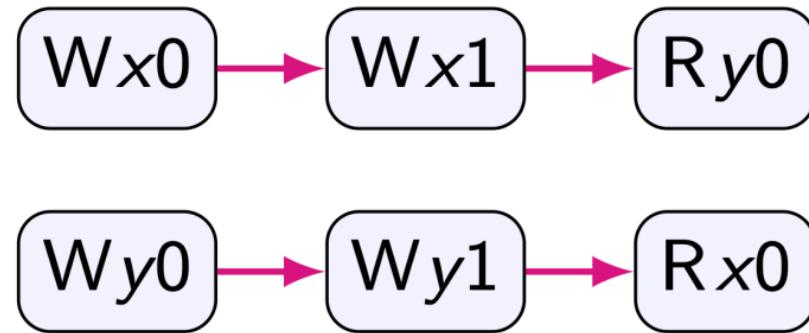
Computation: $\forall R_{xv}$

$R_{xv} < R_{xv}$

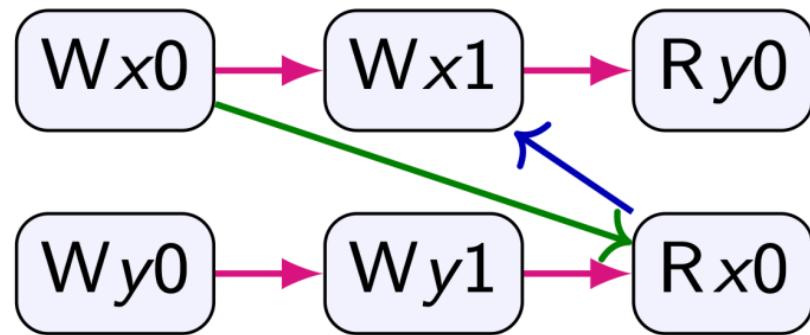
$\forall W_{xu}$ either $W_{xu} \leq W_{xv}$ or $R_{xv} < W_{xu}$

Hardware Reordering?

$x := 0; x := 1; r := y \parallel y := 0; y := 1; s := x$

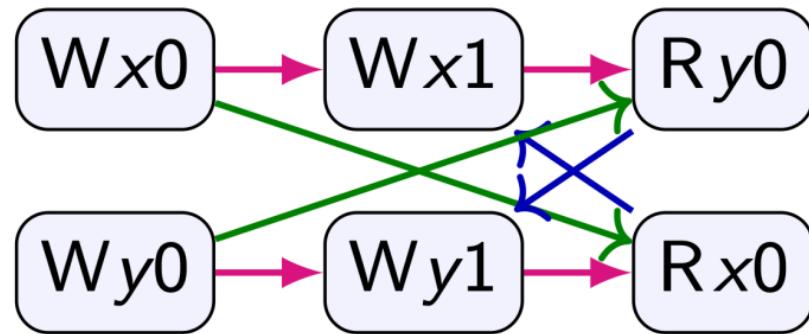


Hardware Reordering?

$$x := 0; x := 1; r := y \parallel y := 0; y := 1; s := x$$


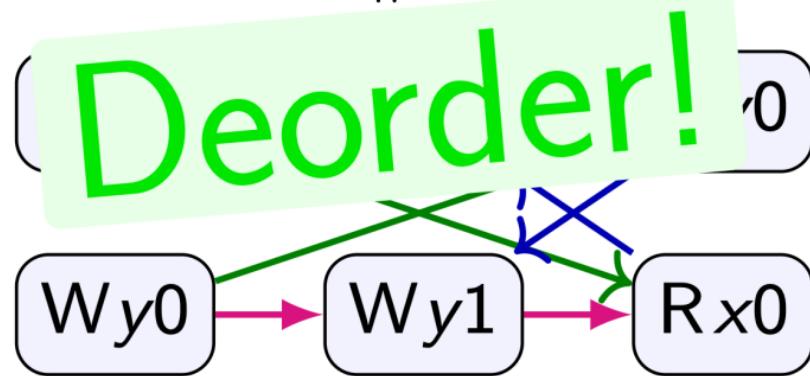
Hardware Reordering?

$x := 0; x := 1; r := y \parallel y := 0; y := 1; s := x$



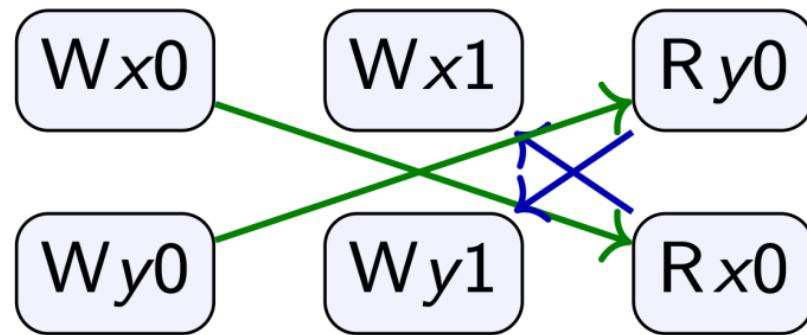
Hardware Reordering?

$x := 0; x := 1; r := y \parallel y := 0; v := 1; s := x$



Deordering!

$x := 0; x := 1; r := y \parallel y := 0; y := 1; s := x$



loring!

$x := 0; x := \dots = y \parallel y := s - 1; s := x$

SC-DRF?

y_0

W_{y1}

lering!

$x := 0; x := s \quad y := y \parallel y := s - 1; s := x$

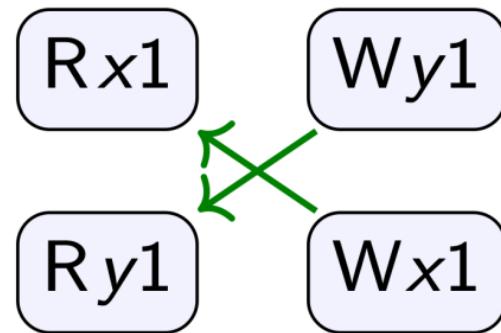
Thin Air?

y_0

W_{y1}

SC-DRF? Thin Air?

$\text{if}(x)\{y:=1\} \parallel \text{if}(y)\{x:=1\}$



SC-DRF? Thin Air?

$\text{if}(x)\{y:=1\} \parallel \text{if}(y)\{x:=1\}$

Dependencies!

Ry1

Wx1

SC-DRF? Thin Air?

Syntactic Dependencies!

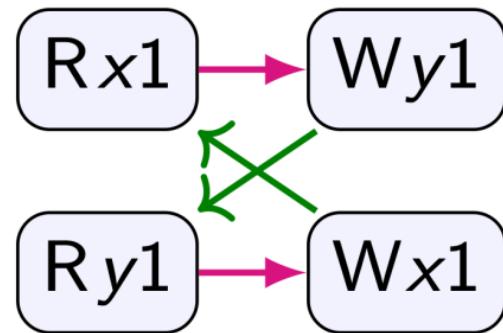
$\cdot \text{if}(x)\{y:=1\} \parallel \text{if}(y)\{x:=1\}$

Ry1

VV[^]A₊

Syntactic Dependencies!

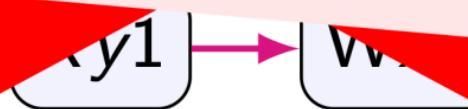
$\text{if}(x)\{y:=1\} \parallel \text{if}(y)\{x:=1\}$



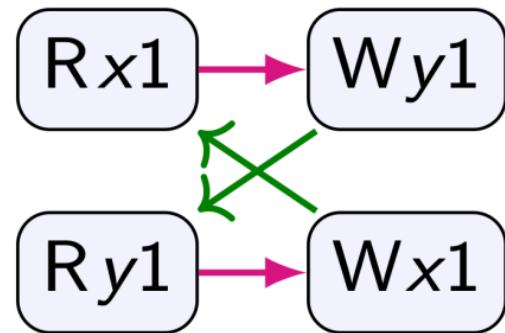
stic Dependencies!

```
1. if(v := 1) || if(v <= 1)
```

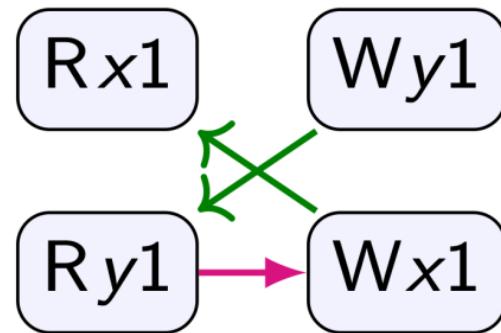
Compiler Reordering?



Compiler Reordering?

$$r := x; y := (r * 0) + 1 \parallel s := y; x := s$$


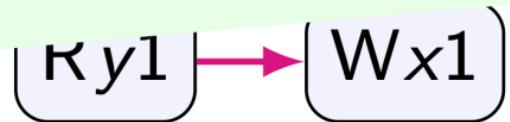
Compiler Reordering?

$$r := x; y := (r * 0) + 1 \parallel s := y; x := s$$


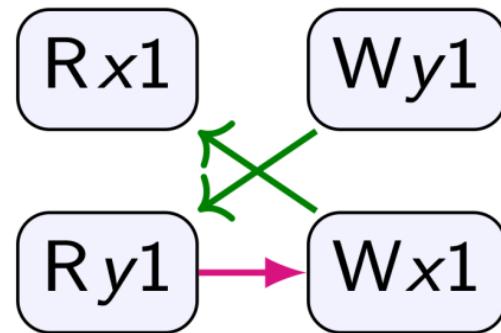
Compiler Reordering?

$$r := x; y := (r * 0) + 1 \parallel s := y; x := s$$

Semantic Dependencies!



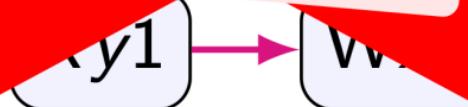
Semantic Dependencies!

$$r := x; y := (r * 0) + 1 \parallel s := y; x := s$$


Semantic Dependencies!

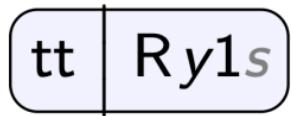
$r := s$ $(r * 0) + 1 \parallel s$ $x := s$

How?



Pomsets with Preconditions

$s := y$



$x := s$



Pomsets with Preconditions

$s := y$

;

$x := s$



Pomsets with Preconditions and Predicate Transformers

$s := y$

;

$x := s$

tt	Ry1s
----	------

s=1	Wx1
-----	-----

D	$\tau^D(\psi)$
\emptyset	ψ
{Ry1s}	$(1=s) \Rightarrow \psi$

E	$\tau^E(\psi)$
\emptyset	ψ
{Wx1}	ψ

Pomsets with Preconditions and Predicate Transformers

$s := y$

;

$x := s$

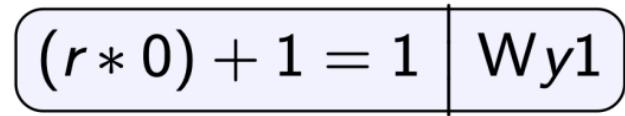


D	$\tau^D(\psi)$
\emptyset	ψ
$\{Ry1s\}$	$(1=s) \Rightarrow \psi$

E	$\tau^E(\psi)$
\emptyset	ψ
$\{Wx1\}$	ψ

Pomsets with Preconditions and Predicate Transformers

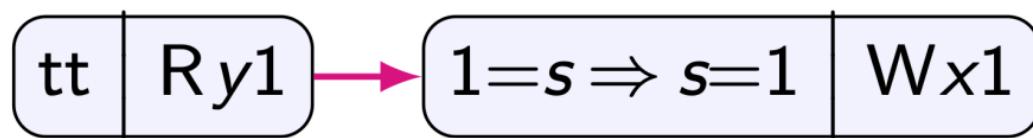
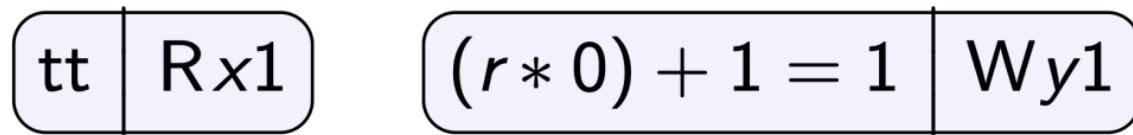
$r := x$; $y := (r * 0) + 1$



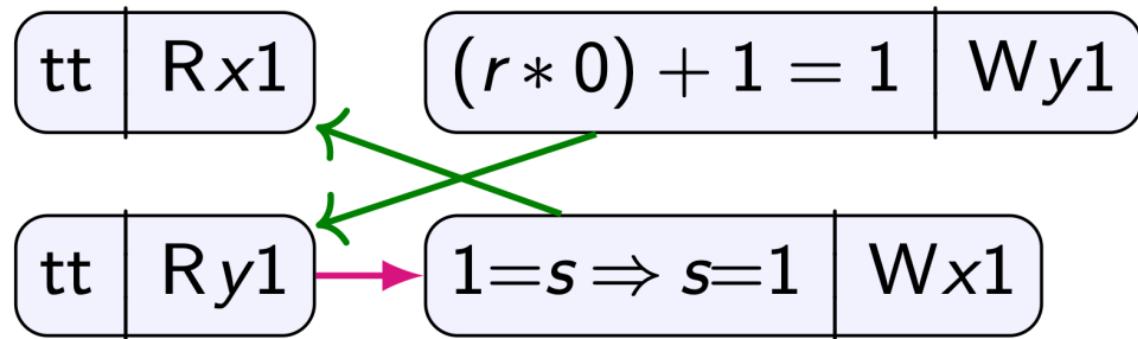
D	$\tau^D(\psi)$
\emptyset	ψ
$\{Rx1r\}$	$(1=r) \Rightarrow \psi$

E	$\tau^E(\psi)$
\emptyset	ψ
$\{Wy1\}$	ψ

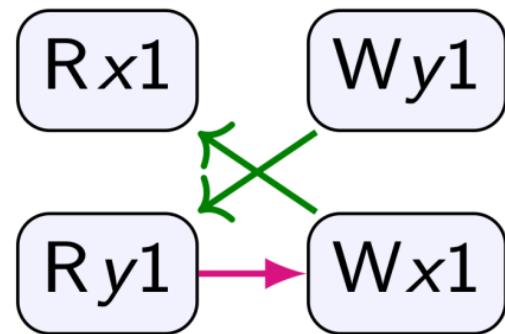
Pomsets with Preconditions and Predicate Transformers

$$r := x; y := (r * 0) + 1 \parallel s := y; x := s$$


Pomsets with Preconditions and Predicate Transformers

$$r := x; y := (r * 0) + 1 \parallel s := y; x := s$$


Pomsets with Preconditions and Predicate Transformers

$$r := x; y := (r * 0) + 1 \parallel s := y; x := s$$


Pomsets with Preconditions and Predicate Transformers

Logic=Local order

Pomset=Other order

Control Dependencies?

Pomelo - Out

Control Independencies?

$r := x; \text{ if } (r \neq 0) \{y := 1\}; \text{ if } (r = 0) \{y := 1\}$

Rx1 r

$r \neq 0$	Wy1
------------	-----

$r = 0$	Wy1
---------	-----

Control Independencies?

$r := x; \text{if } (r \neq 0) \{y := 1\}; \text{if } (r = 0) \{y := 1\}$

Rx1 r

Merging!

0	Wy1
---	-----

Merging!

$r := x; \text{ if } (r \neq 0) \{y := 1\}; \text{ if } (r = 0) \{y := 1\}$

Rx1 r

$r \neq 0 \vee r = 0$ | Wy1

ing!

```
r:=x; if(i>0){v:=1}; i<0){y:=1}
```

Associative?

Associative! (Left to Right)

$r := x$

Rx1 r

$s := y$

Ry1 s

if(s) { $z := r * (s - 1)$ }

$(s \neq 0) \wedge (r * (s - 1)) = 0$ | Wz0

$$\frac{C}{\emptyset} \quad \frac{\tau^C(\psi)}{\psi}$$
$$\{Rx1r\} \quad (1=r) \Rightarrow \psi$$

$$\frac{D}{\emptyset} \quad \frac{\tau^D(\psi)}{\psi}$$
$$\{Ry1s\} \quad (1=s) \Rightarrow \psi$$

$$\frac{E}{\emptyset} \quad \frac{\tau^E(\psi)}{\psi}$$
$$\{Wz0\} \quad \psi$$

Associative! (Left to Right)

$$(r := x \quad ; \quad s := y)$$

Rx1 r

Ry1 s

$$\text{if}(s) \{z := r*(s-1)\}$$

$(s \neq 0) \wedge (r*(s-1)) = 0$ | Wz0

$$\frac{C}{\emptyset} \quad \frac{}{\psi}$$

$$\tau^C(\psi)$$

$$\{Rx1r\} \quad (1=r) \Rightarrow \psi$$

$$\frac{D}{\emptyset} \quad \frac{}{\psi}$$

$$\tau^D(\psi)$$

$$\{Ry1s\} \quad (1=s) \Rightarrow \psi$$

$$\frac{E}{\emptyset} \quad \frac{}{\psi}$$

$$\tau^E(\psi)$$

$$\{Wz0\} \quad \psi$$

Associative! (Left to Right)

$$(r := x \quad ; \quad s := y)$$

Rx1 r

Ry1 s

$$\text{if}(s) \{z := r*(s-1)\}$$

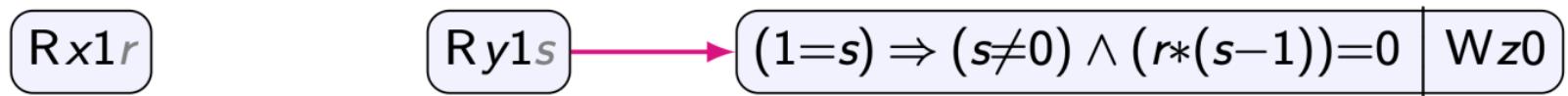
($s \neq 0$) \wedge ($r*(s-1) = 0$) | Wz0

D	$\tau^D(\psi)$
\emptyset	ψ
{Rx1 r }	$(1=r) \Rightarrow \psi$
{Ry1 s }	$(1=s) \Rightarrow \psi$
{Rx1 r , Ry1 s }	$(1=r) \Rightarrow (1=s) \Rightarrow \psi$

E	$\tau^E(\psi)$
\emptyset	ψ
{Wz0}	ψ

Associative! (Left to Right)

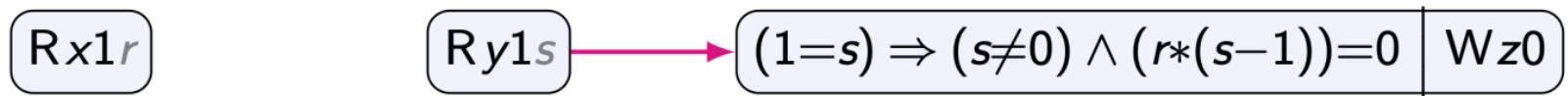
$(r := x ; s := y) ; \text{if}(s)\{z := r * (s - 1)\}$



D	$\tau^D(\psi)$	E	$\tau^E(\psi)$
\emptyset	ψ	\emptyset	ψ
$\{\text{Rx1}r\}$	$(1 = r) \Rightarrow \psi$	$\{\text{Wy1}\}$	ψ
$\{\text{Ry1}s\}$	$(1 = s) \Rightarrow \psi$		
$\{\text{Rx1}r, \text{Ry1}s\}$	$(1 = r) \Rightarrow (1 = s) \Rightarrow \psi$		

Associative! (Left to Right)

$(r := x ; s := y) ; \text{if}(s)\{z := r * (s - 1)\}$



D	$\tau^D(\psi)$
$\emptyset, \{\text{Wy1}\}$	ψ
$\{\text{Rx1}_r\}, \{\text{Rx1}_r, \text{Wy1}\}$	$(1 = r) \Rightarrow \psi$
$\{\text{Ry1}_s\}, \{\text{Ry1}_s, \text{Wy1}\}$	$(1 = s) \Rightarrow \psi$
$\{\text{Rx1}_r, \text{Ry1}_s\}, \{\text{Rx1}_r, \text{Ry1}_s, \text{Wy1}\}$	$(1 = r) \Rightarrow (1 = s) \Rightarrow \psi$

Associative! (Right to Left)

 $r := x$ $\boxed{\text{Rx1}r}$ $s := y$ $\boxed{\text{Ry1}s}$ $\text{if}(s) \{ z := r * (s - 1) \}$ $\boxed{(s \neq 0) \wedge (r * (s - 1)) = 0 \mid \text{Wz0}}$

$$\frac{C}{\emptyset} \quad \boxed{\tau^C(\psi)}$$
$$\{ \text{Rx1}r \} \quad (1 = r) \Rightarrow \psi$$

$$\frac{D}{\emptyset} \quad \boxed{\tau^D(\psi)}$$
$$\{ \text{Ry1}s \} \quad (1 = s) \Rightarrow \psi$$

$$\frac{E}{\emptyset} \quad \boxed{\tau^E(\psi)}$$
$$\{ \text{Wz0} \} \quad \psi$$

Associative! (Right to Left)

$r := x \quad (s := y \quad ; \quad \text{if}(s) \{z := r * (s - 1)\})$

Rx1 r

Ry1 s

$(1 = s) \Rightarrow (s \neq 0) \wedge (r * (s - 1)) = 0$

Wz0

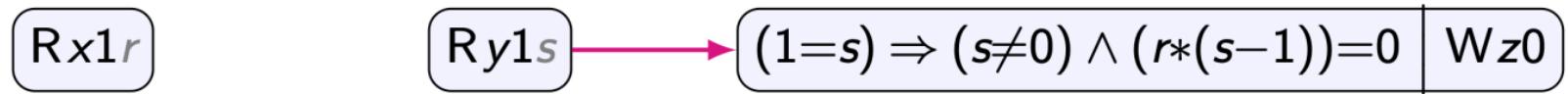
$$\frac{}{\begin{array}{c} C \\ \emptyset \\ \{Rx1r\} \end{array}} \quad \frac{}{\begin{array}{c} \tau^C(\psi) \\ \psi \\ (1=r) \Rightarrow \psi \end{array}}$$

$$\frac{}{\begin{array}{c} D \\ \emptyset \\ \{Ry1s\} \end{array}} \quad \frac{}{\begin{array}{c} \tau^D(\psi) \\ \psi \\ (1=s) \Rightarrow \psi \end{array}}$$

$$\frac{}{\begin{array}{c} E \\ \emptyset \\ \{Wy1\} \end{array}} \quad \frac{}{\begin{array}{c} \tau^E(\psi) \\ \psi \\ \psi \end{array}}$$

Associative! (Right to Left)

$r := x \quad (s := y ; \text{ if } (s) \{z := r * (s - 1)\})$

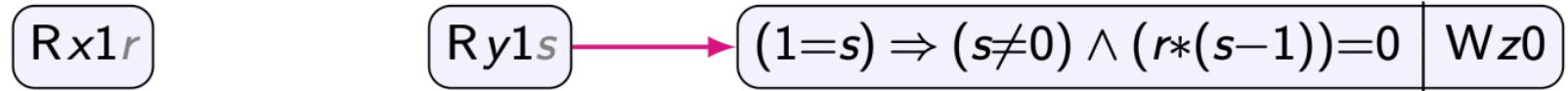


C	$\tau^C(\psi)$
\emptyset	ψ
{Rx1 r }	$(1=r) \Rightarrow \psi$

D	$\tau^D(\psi)$
$\emptyset, \{Wy1\}$	ψ
{Ry1 s }, {Ry1 s , Wy1}	$(1=s) \Rightarrow \psi$

Associative! (Right to Left)

$r := x ; (s := y ; \text{if}(s) \{z := r * (s - 1)\})$

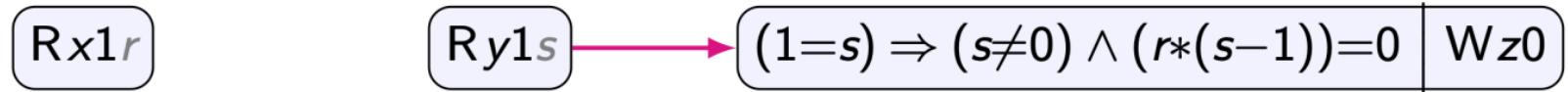


C	$\tau^C(\psi)$
\emptyset	ψ
$\{Rx1r\}$	$(1=r) \Rightarrow \psi$

D	$\tau^D(\psi)$
$\emptyset, \{Wy1\}$	ψ
$\{Ry1s\}, \{Ry1s, Wy1\}$	$(1=s) \Rightarrow \psi$

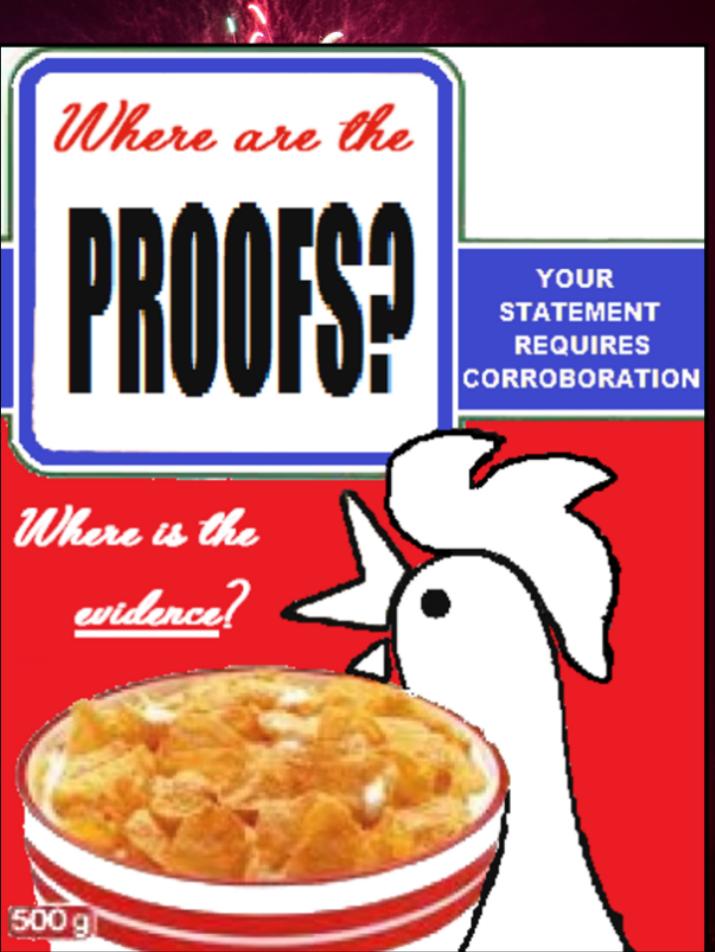
Associative! (Right to Left)

$r := x ; (s := y ; \text{if}(s) \{z := r * (s - 1)\})$



D	$\tau^D(\psi)$
$\emptyset, \{\text{Wy1}\}$	ψ
$\{\text{Rx1}_r\}, \{\text{Rx1}_r, \text{Wy1}\}$	$(1=r) \Rightarrow \psi$
$\{\text{Ry1}_s\}, \{\text{Ry1}_s, \text{Wy1}\}$	$(1=s) \Rightarrow \psi$
$\{\text{Rx1}_r, \text{Ry1}_s\}, \{\text{Rx1}_r, \text{Ry1}_s, \text{Wy1}\}$	$(1=r) \Rightarrow (1=s) \Rightarrow \psi$





```

Lemma seq_assoc2 {a : thread_id} s1 s2 s3 :
Semantics a (Stmt.seq s1 (Stmt.seq s2 s3)) ≤1
Semantics a (Stmt.seq (Stmt.seq (Stmt.seq s1 s2) s3)).
```

Proof.

```

(* dep P ⊑ dep P1 ∪ dep P2 ∪ (events_set P1 × events_set P2)*)
```

```

assert ((forall {P P1 P2} (SEQ12 : SEQ P P1 P2) 'dep P))
      /\ dep P1 = restr_rel (events_set P1) 'dep P)
      /\ dep P2 = restr_rel (events_set P2) 'dep P))
```

```

{ split; ins.
  rewrite dep_restr1; [done | by apply SEQ12].
  intros P0 P0.
  assert (wf P0) as WF by eby eapply semantics_wf.
  rename s0 into s1, P2 into P23.
  inv interp2.
  rename s0 into s2, s4 into s3.■
```

```

assert (events_set P23 = events_set P2 ∪ events_set P3) as EVENTS23
```

```

{ apply PE0.
  assert (wf P1 ∩ wf P2 ∩ wf P3) as [WF1 [WF2 WF3]].
```

```

{ splits; eapply semantics_wf; eauto. }
```

```

assert (forall d (DD : events_set P23 d) as SAMELBL_23_0.
  { symmetry. apply (seq_label_union PE); auto. })
```

```

assert (forall d (DD : events_set P23 d) as SAMELBL_23_1.
  { apply (P23 d = P0 d) as SAMELBL_23_0.
    splits; eapply semantics_wf; eauto. })
```

```

assert (forall d (DD : events_set P23 d) as SAMELBL_23_2.
  { apply (P23 d = P0 d) as SAMELBL_23_0.
    { symmetry. apply (seq_label_union PE0); auto. } })
```

```

assert (forall d (DD : events_set P23 d) as SAMELBL_23_0.
  { ins. etransitivity; [| apply SAMELBL_23_0. ]})
  symmetry. apply (seq_label_union PE0); auto. }
```

```

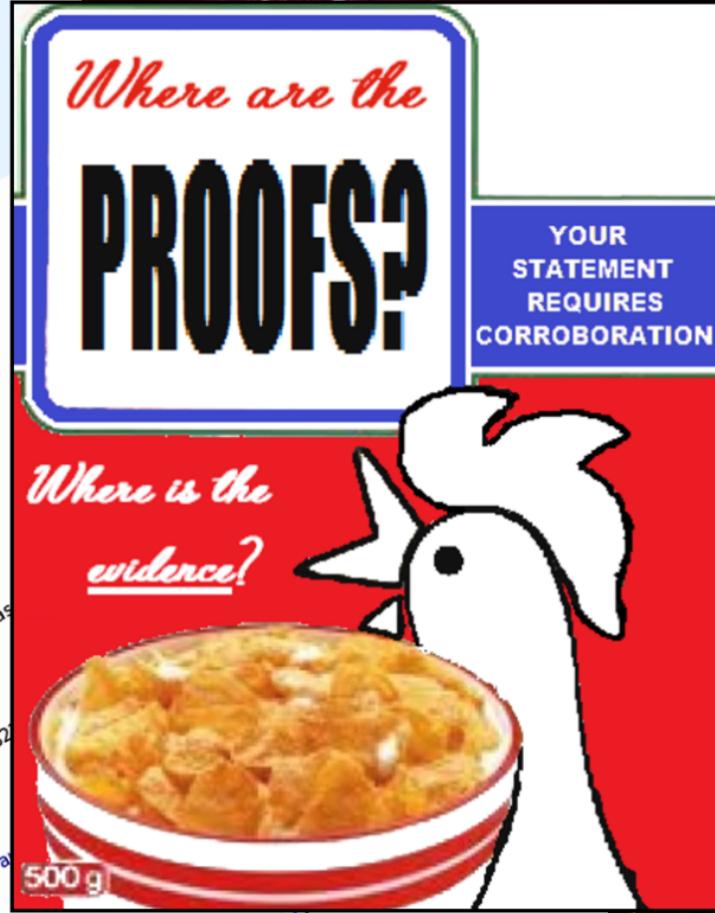
assert (forall d (DD : events_set P23 d) as SAMELBL_23_0.
  { P3 d = P0 d as SAMELBL_23_0.
    etransitivity; [| apply SAMELBL_23_0. ]})
  symmetry. apply (seq_label_union PE0); auto. }
```

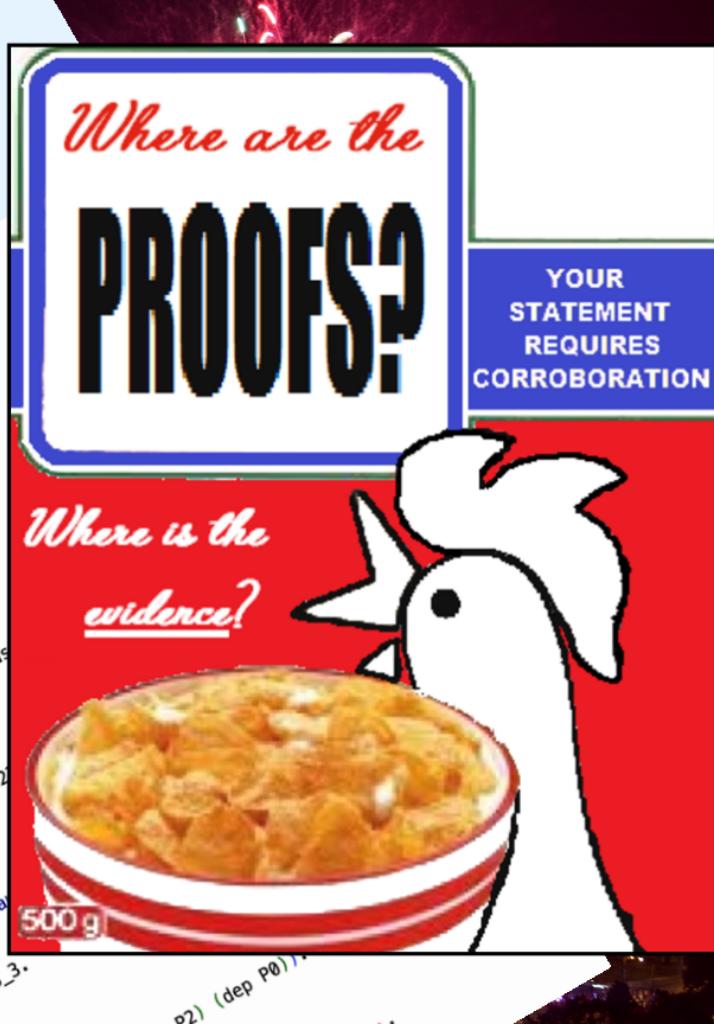
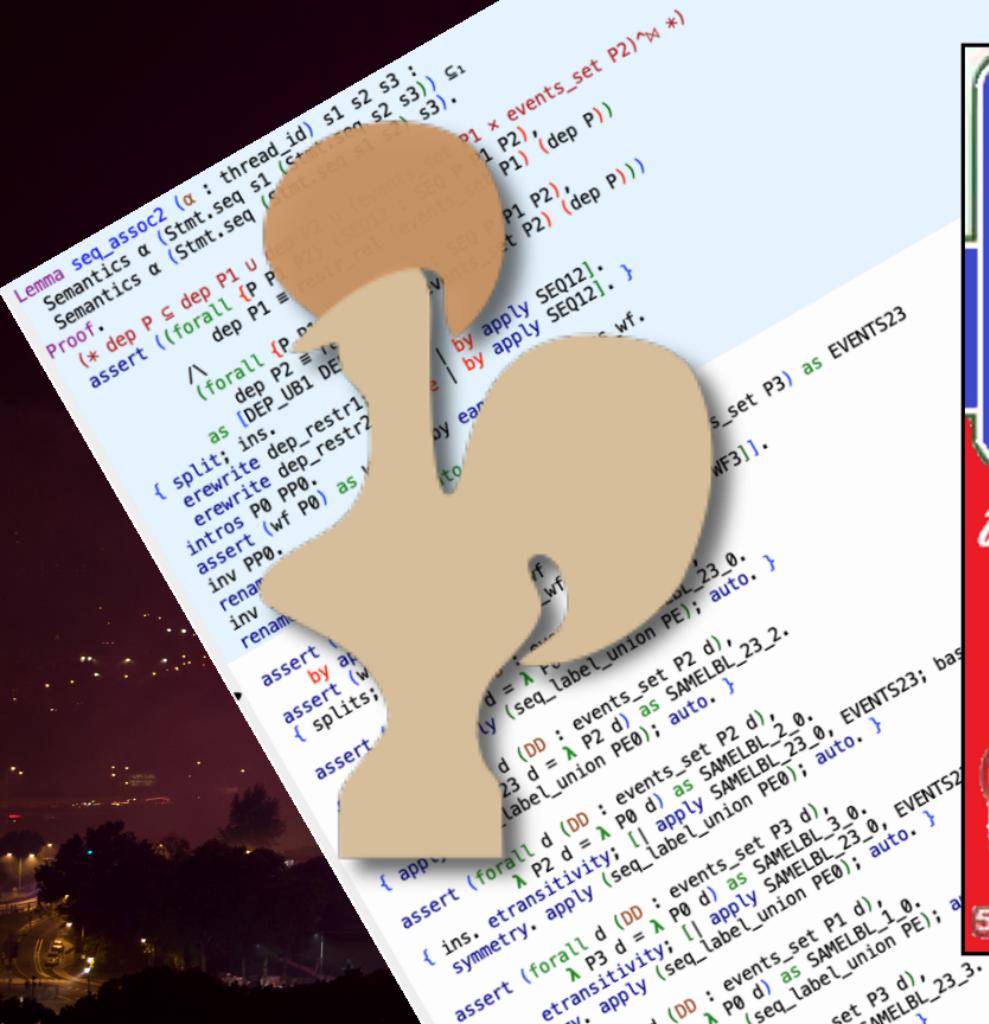
```

assert (forall d (DD : events_set P23 d) as SAMELBL_23_0.
  { P3 d = P0 d as SAMELBL_23_0.
    etransitivity; [| apply SAMELBL_23_0. ]})
  symmetry. apply (seq_label_union PE0); auto. }
```

```

assert (forall d (DD : events_set P23 d) as SAMELBL_23_3.
  { P3 d = P0 d as SAMELBL_23_3.
    etransitivity; [| apply SAMELBL_23_3. ]})
  symmetry. apply (seq_label_union PE0); auto. }
```





Real Hardware?



Multicopy Atomic Architectures

$S_1 ; S_2$

Let order include (in addition to dependency):

Coherence ($Wx \rightarrow Wx$)

Release/Acquire Access/Fences ($Wx \rightarrow W^{\text{rel}}y, R^{\text{acq}}x \rightarrow Wy$)

SC Access/Fences ($W^{\text{sc}}x \rightarrow R^{\text{sc}}y$)

Multicopy Atomic Architectures

$S_1 ; S_2$

Let order include (in addition to dependency):

Coherence ($Wx \rightarrow Wx$)

Release/Acquire Access/Fences ($Wx \rightarrow W^{\text{rel}}y, R^{\text{acq}}x \rightarrow Wy$)

SC Access/Fences ($W^{\text{sc}}x \rightarrow R^{\text{sc}}y$)

Arm8 optimal for: Relaxed ✓ Release ✓ Acquiring ✗

Multicopy Atomic Architectures

$S_1 ; S_2$

Let order include (in addition to dependency):

Coherence ($Wx \rightarrow Wx$)

Release/Acquire Access/Fences ($Wx \rightarrow W^{\text{rel}}y, R^{\text{acq}}x \rightarrow Wy$)

SC Access/Fences ($W^{\text{sc}}x \rightarrow R^{\text{sc}}y$)

Arm8 optimal for: Relaxed ✓ Release ✓ Acquiring ✗

Arm8 some work: Relaxed ✓ Release ✓ Acquiring ✓

Multicopy Atomic Architectures

Armed Cats: Formal Concurrency Modelling at Arm

JADE ALGLAVE, Arm Ltd and University College London
WILL DEACON and RICHARD GRISENTHWAITE, Arm Ltd

ANTOINE HACQUARD, EPITA Research and Development Laboratory
LUC MARANGET, INRIA

We report on the process for formal concurrency modelling at Arm. An initial formal consistency model of the Arm architecture, written in the cat language, was published and upstreamed to the herd+diy tool suite in 2017. Since then, we have extended the original model with extra features, for example, mixed-size accesses, and produced two provably equivalent alternative formulations.

In this article, we present a comprehensive review of work done at Arm on the consistency model. Along the way, we also show that our principle for handling mixed-size accesses applies to x86: We confirm this via vast experimental campaigns. We also show that our alternative formulations are applicable to any model phrased in a style similar to the one chosen by Arm.

CCS Concepts: • Hardware → Hardware test; • Computer systems organization → Architectures; • Software and its engineering → Software organization and properties; • Theory of computation → Semantics and reasoning

Additional Key Words and Phrases: Concurrency Weak memory models, arm architecture, linux, mixed-size accesses

ACM Reference format:
Jade Alglave, Will Deacon, Richard Grisenthwaite, Antoine Hacquard, and Luc Maranget. 2021. Armed Cats: Formal Concurrency Modelling at Arm. ACM Trans. Program. Lang. Syst. 43, 2, Article 8 (July 2021), 54 pages.
<https://doi.org/10.1145/3458926>

1 INTRODUCTION

Arm has invested in formal modelling of the concurrency aspects of its architecture, which led to the publication of a formal model in 2017 [23]. This model has since been maintained and enhanced in various ways, for example, by adding features such as mixed-size accesses [9]. The importance and necessity of that mixed-size extension can be seen in the fact that without it Linux's lock structure cannot be used soundly on Arm machines, as we detail in this article.

Arm has also developed and released two alternative formulations [1]. All three models written in the cat language [19], a domain-specific language for writing formal consistency models in a concise and executable manner. The original model [23] is written in idiomatic cat, as

Let order include (in addition to dependency)

Coherence ($Wx \rightarrow Wx$)

Release/Acquire Access/Fences ($Wx \rightarrow Rx$)

SC Access/Fences ($W^{sc}x \rightarrow R^{sc}y$)

Arm8 optimal for: Relaxed ✓ Release ✓ Acquire ✓

Arm8 some work: Relaxed ✓ Release ✓ Acquire ✓

Multiprocessor Atomic Architectures

Logic=Local order
Pomset=Other order

, - success ($\forall\forall^{\sim}x \rightarrow R^{sc}y$)
Arm8 optimal for: Relaxed ✓ Release ✓ Acquire ✓
Arm8 some work: Relaxed ✓ Release ✓ Acquire ✓

Formal Concurrency Modelling at Arm
<https://doi.org/10.1145/3373453.3373500>

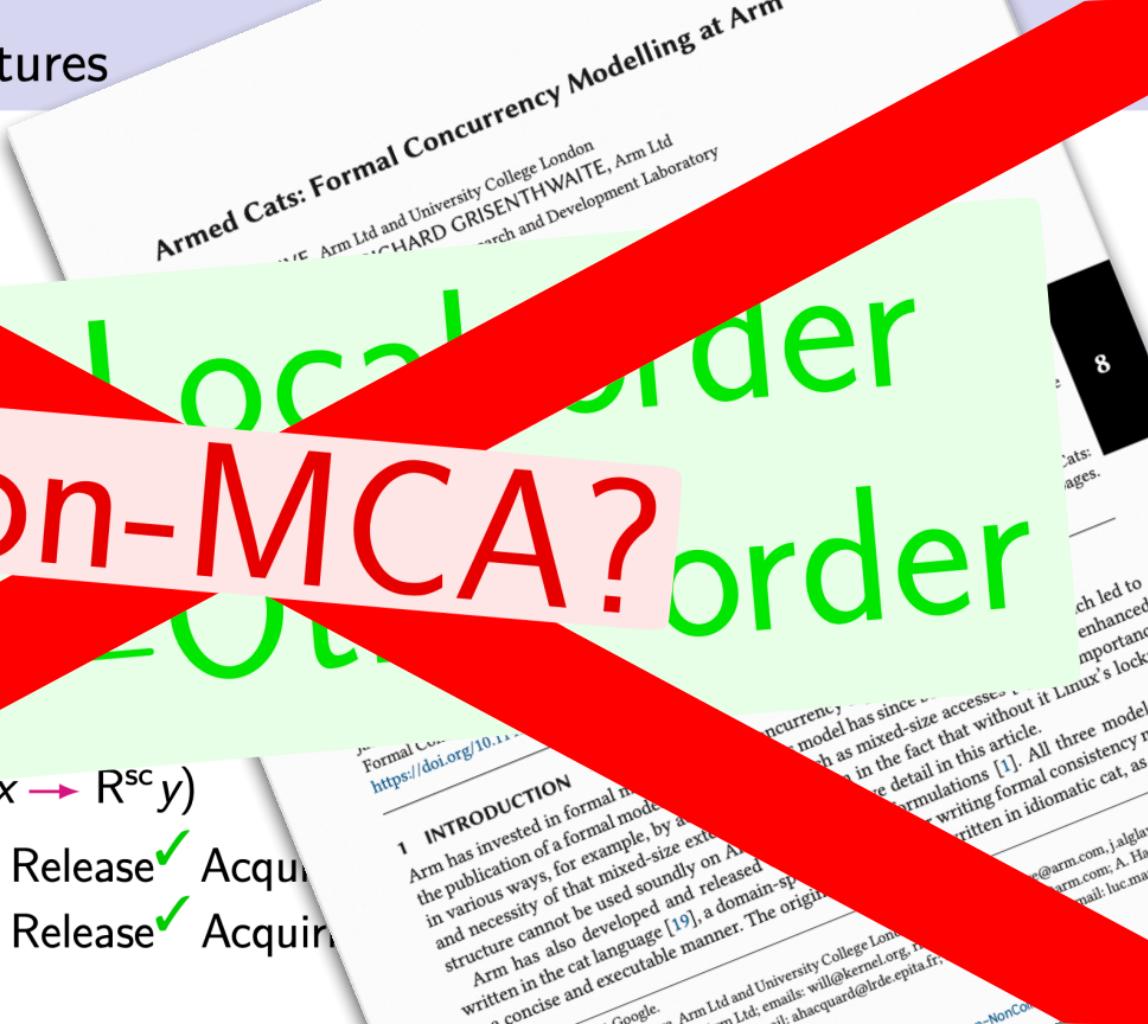
1 INTRODUCTION

Arm has invested in formal modelling of the concurrency model since the publication of a formal model in 2017 [23]. This model has since been enhanced in various ways, for example, by adding features such as mixed-size accesses and necessity of that mixed-size extension can be seen in the fact that without it Linux's lock structure cannot be used soundly on Arm machines, as we detail in this article. Arm has also developed and released two alternative formulations [1]. All three models written in the cat language [19], a domain-specific language for writing formal consistency models in a concise and executable manner. The original model [23] is written in idiomatic cat, as Google, Arm Ltd and University College London; emails: will@kernel.org, richard.grisenthwaite@arm.com, jadalgave@arm.com, jadalgave@lri.fr, ahacquard@lri.fr, L. Marange, INRIA; email: luc.marange@inria.fr

Copy Atomic Architectures

Log Non-MCA?order Pomset Order

Accesses ($\forall v \sim x \rightarrow R^{sc} y$)
A lock is valid for: Relaxed ✓ Release ✓ Acquire ✓
After some work: Relaxed ✓ Release ✓ Acquire ✓



Dependencies for RC11

Replace No-TAR axiom

$sb \cup rf$ is acyclic

Dependencies for RC11

Replace No-TAR axiom

$sb \cup rf$ is acyclic

$sdep \cup rf$ is acyclic

$sdep$ = pomset order

Dependencies for RC11

Replace No-TAR axiom

sb

$$\begin{aligned} \text{rb} &\triangleq \text{rf}^{-1}; \text{mo} \\ \text{eco} &\triangleq (\text{rf} \cup \text{mo} \cup \text{rb})^+ \quad (\text{extended coherence order}) \\ \text{rs} &\triangleq [\text{W}; \text{sb}]_{\text{loc}}?; [\text{W} \sqsupseteq \text{rlx}]; (\text{rf}; \text{rmw})^* \quad (\text{release sequence}) \\ \text{sw} &\triangleq [\text{E} \sqsupseteq \text{rel}]; ([\text{F}]; \text{sb})?; \text{rs}; \text{rf}; \quad (\text{synchronizes with}) \\ &\quad [\text{R} \sqsupseteq \text{rlx}]; (\text{sb}; [\text{F}])?; [\text{E} \sqsupseteq \text{acq}] \\ \text{hb} &\triangleq (\text{sb} \cup \text{sw})^+ \quad (\text{happens-before}) \end{aligned}$$

sdep

$$\begin{aligned} \text{sb}|_{\neq \text{loc}} &\triangleq \text{sb} \setminus \text{sb}|_{\text{loc}} \\ \text{scb} &\triangleq \text{sb} \cup \text{sb}|_{\neq \text{loc}}; \text{hb}; \text{sb}|_{\neq \text{loc}} \cup \text{hb}|_{\text{loc}} \cup \text{mo} \cup \text{rb} \\ \text{psc}_{\text{base}} &\triangleq ([\text{E}^{\text{sc}}] \cup [\text{F}^{\text{sc}}]; \text{hb}?); \text{scb}; ([\text{E}^{\text{sc}}] \cup \text{hb}?; [\text{F}^{\text{sc}}]) \\ \text{psc}_{\text{F}} &\triangleq [\text{F}^{\text{sc}}]; (\text{hb} \cup \text{hb}; \text{eco}; \text{hb}); [\text{F}^{\text{sc}}] \\ \text{psc} &\triangleq \text{psc}_{\text{base}} \cup \text{psc}_{\text{F}} \end{aligned}$$

- $\text{hb}; \text{eco}?$ is irreflexive.
- $\text{rmw} \cap (\text{rb}; \text{mo}) = \emptyset$.
- psc is acyclic.
- $\text{sb} \cup \text{rf}$ is acyclic.

(COHERENCE)
(ATOMICITY)
(SC)
(NO-THIN-AIR)

clic

clic

sdep = pomset order

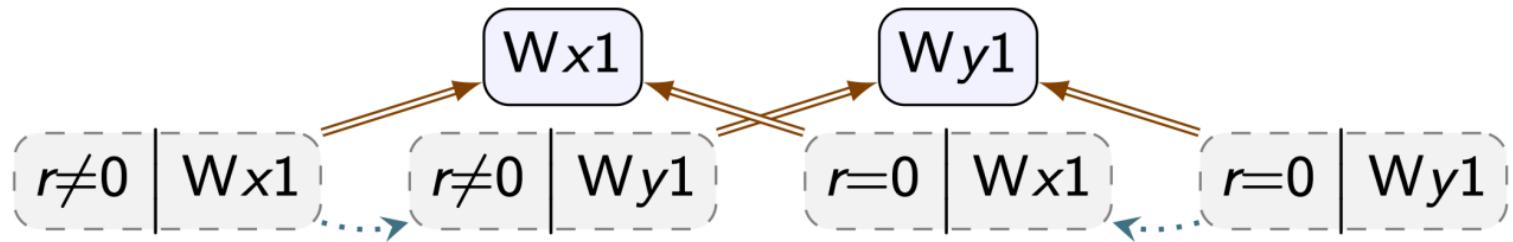
Merging and Program Order?

```
if( $r$ ){ $x := 1$ ;  $y := 1$ } else { $y := 1$ ;  $x := 1$ }
```



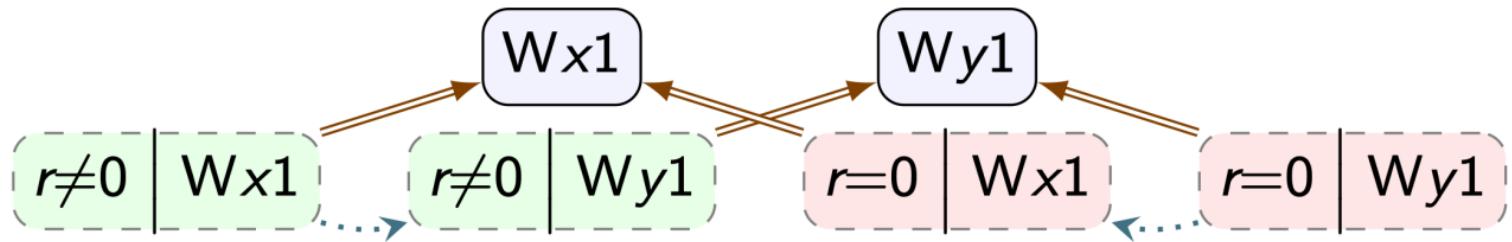
Merging and Program Order?

```
if(r){x:=1; y:=1} else {y:=1; x:=1}
```



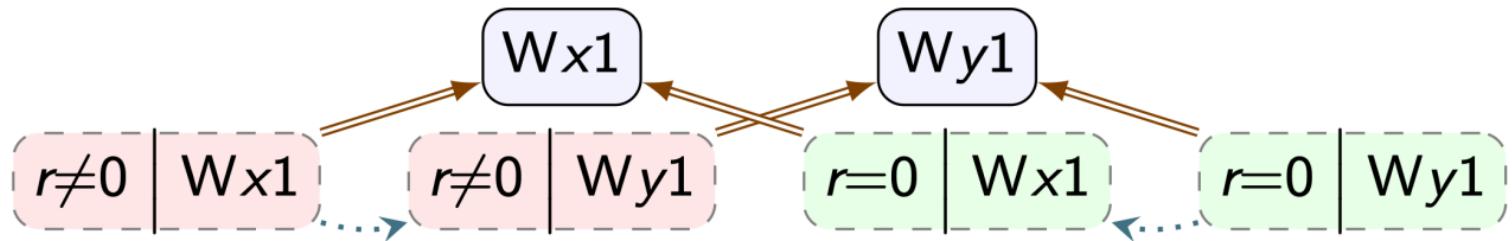
Merging and Program Order?

```
if(r){x:=1; y:=1} else {y:=1; x:=1}
```



Merging and Program Order?

```
if(r){x:=1; y:=1} else {y:=1; x:=1}
```



Merging and Program Order?

The terminal window shows the following content:

```
$ cat data/tests/jctc/jctc1.lit
name=JCTC1
values={0,1}
comment "Should be allowed"
%%%
if(r) {
    r1 := x;
    if (r1 ≥ 0) {
        y := 1
    } else { skip }
} ||| {
    r2 := y;
    x := r2
}
%%%
allow (r1 = 1 && r2 = 1) [] "Allowed, since interthread compiler analysis could
determine that x and y are always non-negative, allowing simplification of r1 ≥
0 to true, and allowing write y = 1 to be moved early."
$ ./pomsets.exe --check --complete data/tests/jctc/jctc1.lit
Allowed, since interthread compiler analysis could determine that x and y are al
ways non-negative, allowing simplification of r1 ≥ 0 to true, and allowing writ
e y = 1 to be moved early. (pass)
$
```

On the left, under the condition $r \neq 0$, the code is shown with annotations: $r1 := x;$, $y := 1$, and $x := r2$. On the right, under the condition $r = 0$, the code is shown with annotations: $r2 := y$ and $x := r2$. Above the code, the expression $x := 1$ is shown.

Merging and Program Order?

if(r) {

$r \neq 0$

Test	PwT-C11	MRD	MRD _{IMM}	MRD _{C11}
TC1	✓	✓	✓	✓
TC2	✓	✓	✓	✓
TC3	✓	✓	✓	✓
TC4	✓	✓	✓	✓
TC5	✓	✓	✓	✓
TC6	✓	✓	✓	✓
TC7	✓	✓	✓	✓
TC8	✓	✓	✓	✓

```
$ cat data/tests/jctc/jctc1.lit
name=JCTC1
values={0,1}
comment "Should be allowed"
%%%
{
```

```
    r1 :=
```

Test	PwT-C11	MRD	MRD _{IMM}	MRD _{C11}
TC9	✓	✓	✓	✓
TC10	✓	✓	✓	✓
TC11	⊥	—	—	—
TC12	⊥	✓	✓	✓
TC13	✓	✗	✓	✗
TC17	✓	✗	✓	✗
TC18	✓	✗	✓	✗

$:= 1 \}$

$r = 0 \mid Wy_1$

analysis could be moved early.

thread compiler analysis could determine that x and y are all negative, allowing simplification of $r1 \geq 0$ to true, and allowing writ

\$

Sequential composition for an optimized concurrent language

Dependency = Pomset order = Order seen by other threads

Sequential composition for an optimized concurrent language

Dependency = Pomset order = Order seen by other threads

Very compositional for MCA

Solution for Out-Of-Thin-Air models (C, JavaScript)

Sequential composition for an optimized concurrent language

Dependency = Pomset order = Order seen by other threads

Very compositional for MCA

Solution for Out-Of-Thin-Air models (C, JavaScript)

Denotational semantics

Tractable notion of refinement for peephole optimization

Understandable by human beings

Sequential composition for an optimized concurrent language

Dependency = Pomset order = Order seen by other threads

Very compositional for MCA

Solution for Out-Of-Thin-Air models (C, JavaScript)

Denotational semantics

Tractable notion of refinement for peephole optimization

Understandable by human beings

Main complexity

Calculating dependencies

Merging and unmerging

In conditionals, single execution uses both sides

Sequential composition for an optimized concurrent language

Dependency = Pomset order = Order seen by other threads

Very compositional for MCA

Solution for Out-Of-Thin-Air models (C, JavaScript)

Denotational semantics

Tractable notion of refinement for peephole optimization

Understandable by human beings

Main complexity

Calculating dependencies

Merging and unmerging

In conditionals, single execution uses both sides

More in the paper

RMWs, address calculation, Indirect dependencies, If-introduction, etc

Sequential composition for an optimized concurrent language

Dependency = Pomset order = Order seen by other threads

Tractable notion of consistency
for MCA

Concurrency is easy!

Tractable notion of consistency

Understandable by human beings

Main complexity

Calculating dependencies

Merging and unmerging

In conditionals, single execution uses both sides

More in the paper

RMWs, address calculation, Indirect dependencies, If-introduction, etc

Sequential composition for an optimized concurrent language

Dependency = Pomset order = Order seen by other threads

Implementation for MCA

Concurrency is easy!

Tractable notion of correctness

Understandable by human beings

Sequentiality is hard!

Two-sidedness, single execution uses both sides

More in the paper

RMWs, address calculation, Indirect dependencies, If-introduction, etc