

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

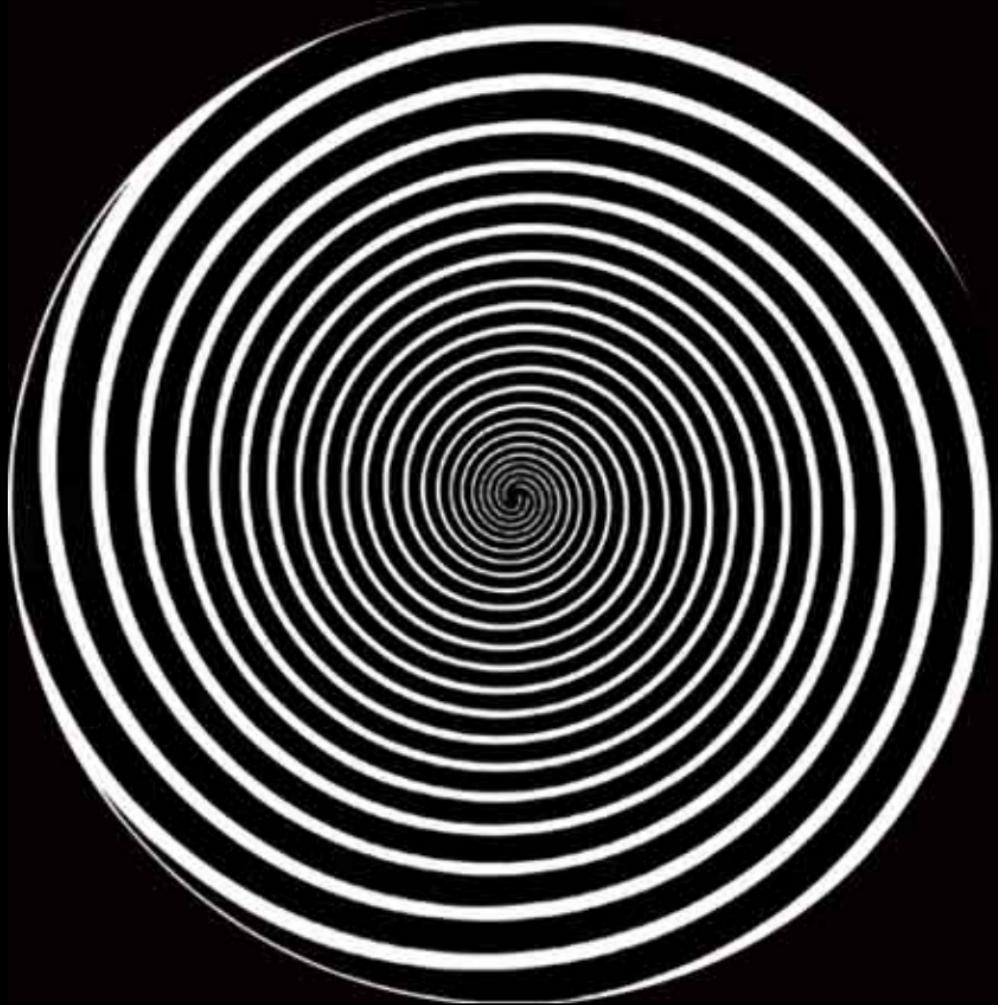
Compositional Semantic Dependencies for Relaxed-Memory Concurrency

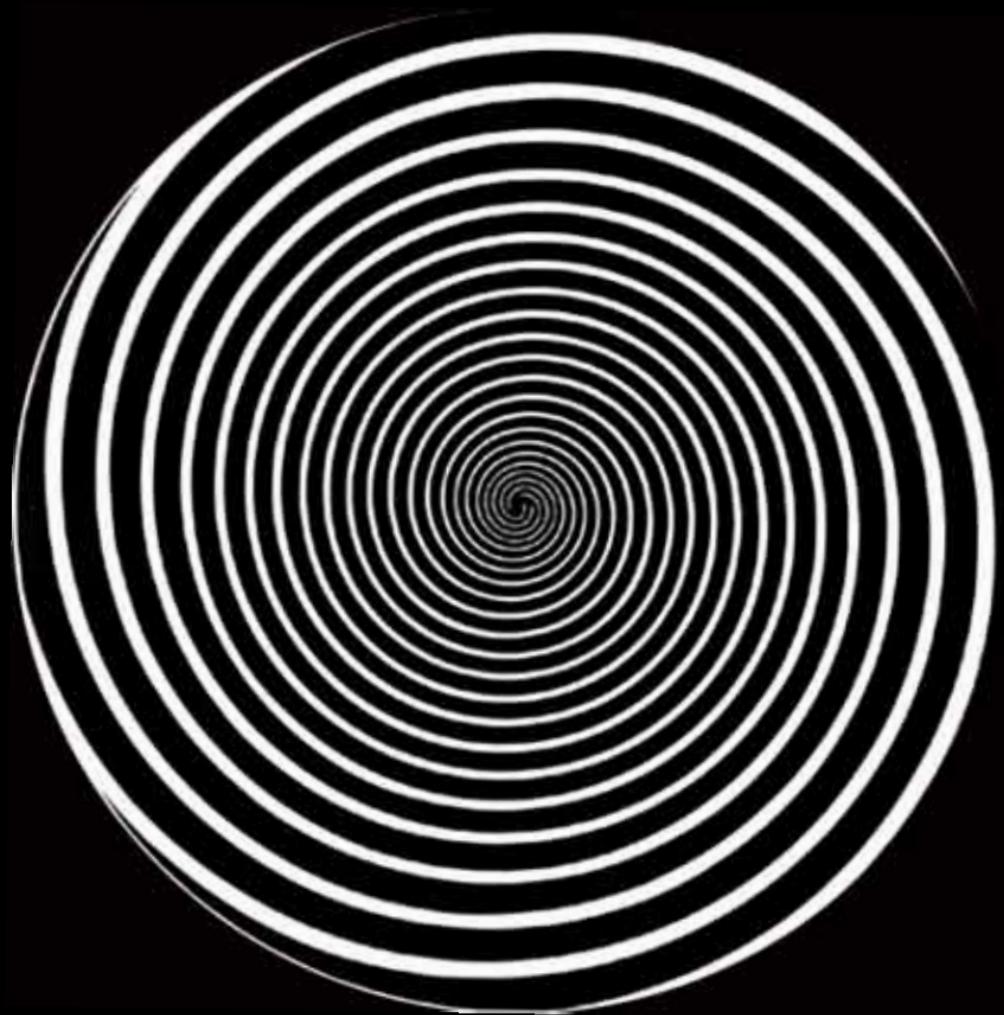
Alan Jeffrey* James Riely† Mark Batty‡
Simon Cooksey† Ilya Kaysin** Anton Podkopaev††

* Roblox † DePaul ‡ Kent

** JetBrains + Cambridge †† HSE

POPL
January 2022





Revert to 1999 ...

Revert to 1999 ...

A model of imperative programming

Revert to 1999 ...

A model of imperative programming

Shared-memory concurrency

Revert to 1999 ...

A model of imperative programming

Shared-memory concurrency

Compositional for both parallelism and sequencing

Revert to 1999 ...

A model of imperative programming

Shared-memory concurrency

Compositional for both parallelism and sequencing

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

Revert to 1999 ...

A model of imperative progr

Shared-memory concu

Compositional for botl

A few other things:

Features: pointer

Optimizations: rea

Java Causality Te

DRF-SC, No-Thir

Efficient impleme

Connection to C1

Automatic litmus

Certified proofs



ces, RMWs

ant read elimination, etc.

g

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!

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

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

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

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

```
-0; r:=x; y:=r
```

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

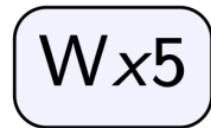
Rx5

Wy6

Wx5

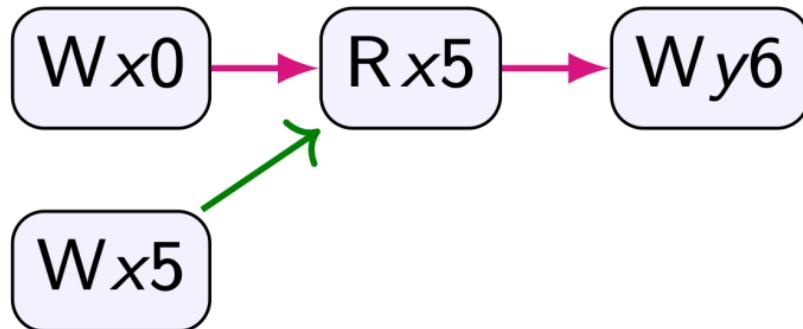
Concurrent Computation: Pomsets! (aka labeled partial orders)

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



Concurrent Computation: Pomsets! (aka labeled partial orders)

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

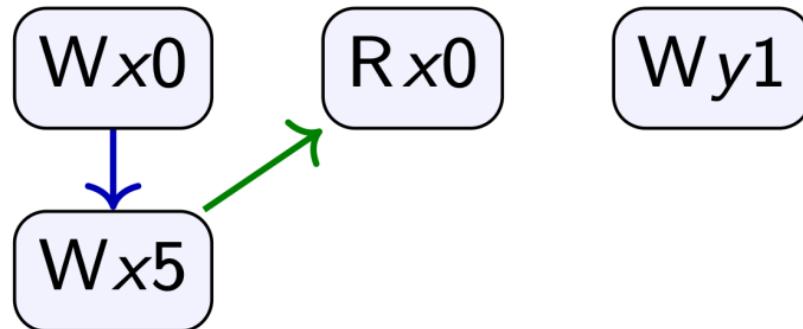


Rxv is *fulfilled* if:

$$\exists Wxv < Rxv$$

Concurrent Computation: Pomsets! (aka labeled partial orders)

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

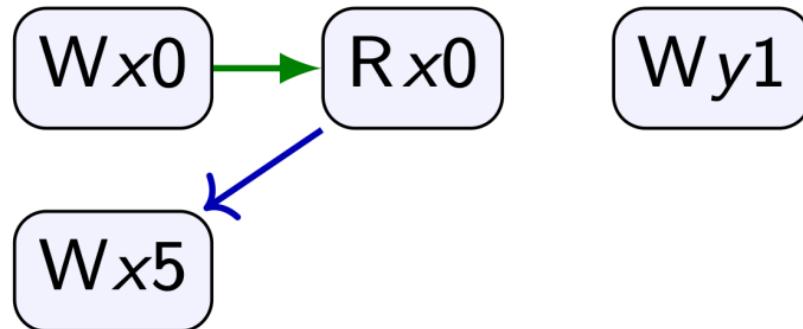


Rxv is *fulfilled* if:

$$\exists Wxv < Rxv$$

$$\forall Wxu \text{ either } Wxu \leqslant Wxv \text{ or } Rxv < Wxu$$

Concurrent Computation: Pomsets! (aka labeled partial orders)

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


R_{xv} is *fulfilled* if:

$$\exists W_{xv} < R_{xv}$$

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

$x := r; \quad y := x; \quad y := r + 1$

Hardware Reordering?

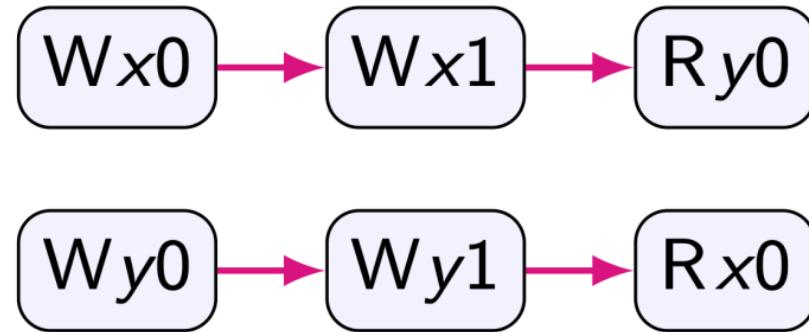
R \leq S if:

$$Rxv < Rxv$$

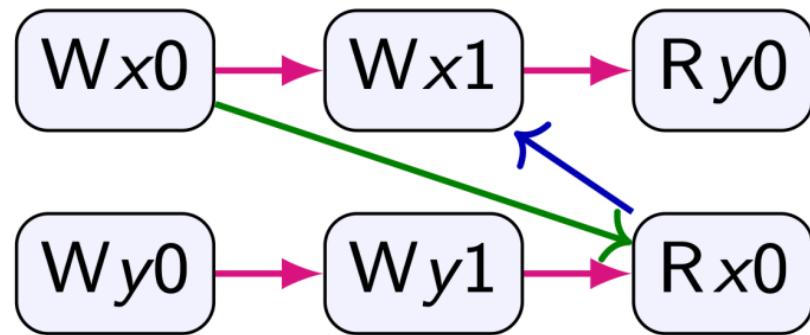
$$\forall Wxu \text{ either } Wxu \leq Wxv \text{ or } Rxv < Wxu$$

Hardware Reordering?

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

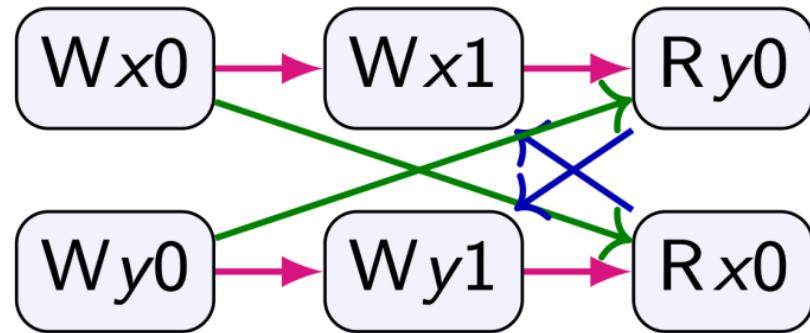


Hardware Reordering?

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


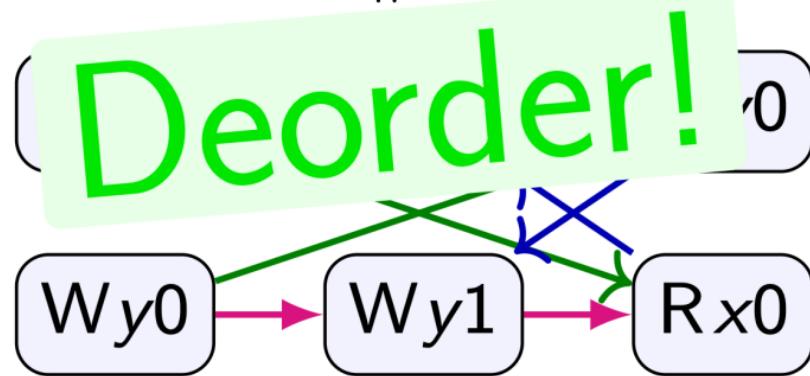
Hardware Reordering?

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

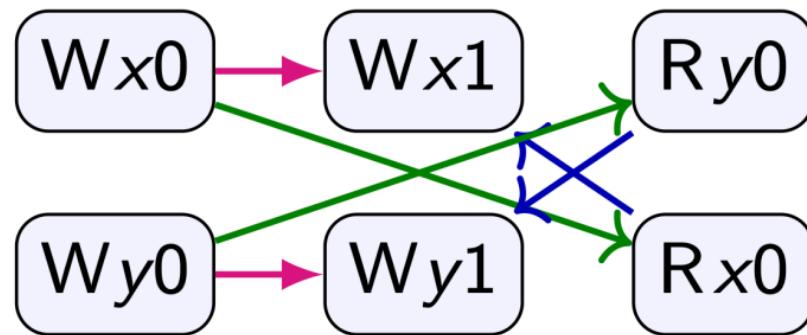


Hardware Reordering?

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



Deordering!

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


lering!

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

SC-DRF?

$y_0 \rightarrow Wy_1$

lering!

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

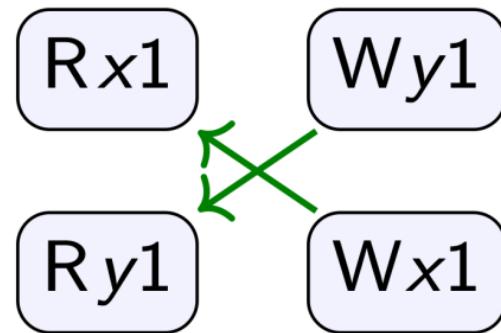
Thin Air?

y_0

Wy1

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!

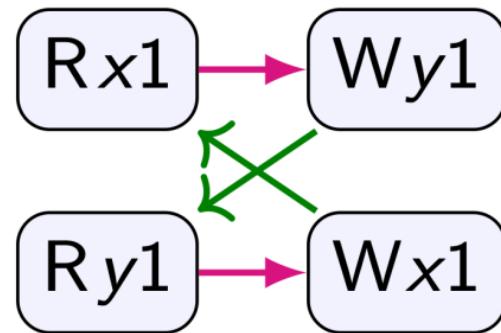
$\cdot \text{if}(x)\{y:=1\} \parallel \text{if}(y)\{x:=1\}$

Ry1

VV[^]A₊

Syntactic Dependencies!

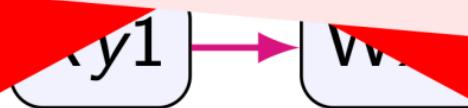
$\text{if}(x)\{y:=1\} \parallel \text{if}(y)\{x:=1\}$



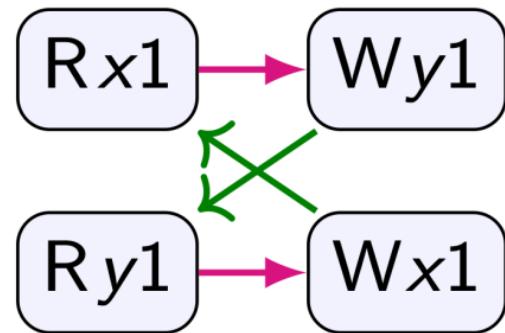
stic Dependencies!

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

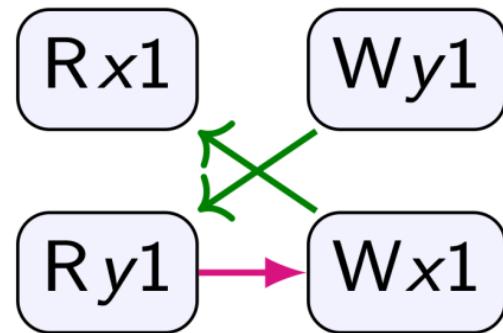
Compiler Reordering?



Compiler Reordering?

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


Compiler Reordering?

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


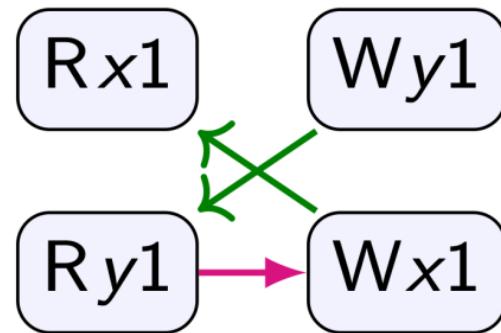
Compiler Reordering?

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

Semantic Dependencies!



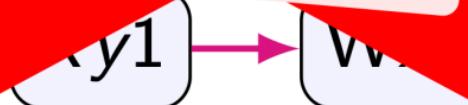
Semantic Dependencies!

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


Semantic Dependencies!

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

How?

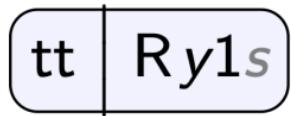


antic Dependencies!

Pomsets
+ Preconditions!

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

$s := y$

;

$x := s$

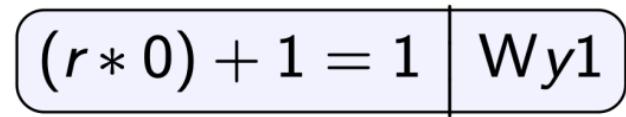
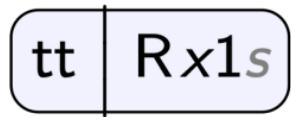


D	$\tau^D(\psi)$
\emptyset	ψ
$\{Ry1s\}$	$(1=s) \Rightarrow \psi$

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

Pomsets with Preconditions and Predicate Transformers

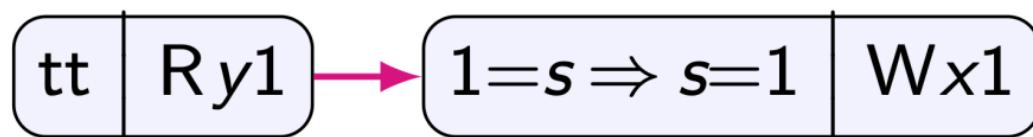
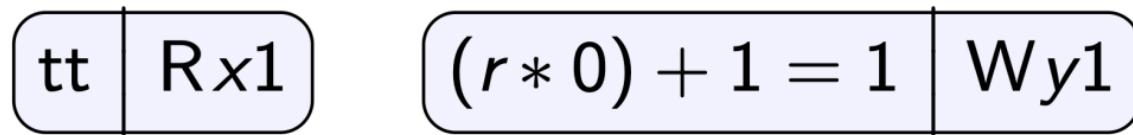
$r := x$; $y := (r * 0) + 1$



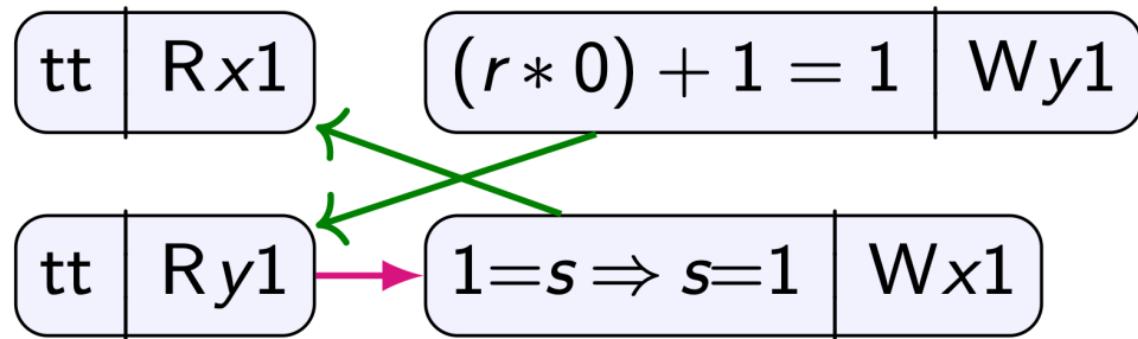
D	$\tau^D(\psi)$
\emptyset	ψ
$\{Rx1r\}$	$(1=r) \Rightarrow \psi$

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

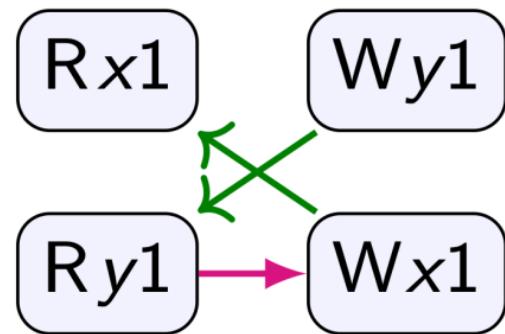
Pomsets with Preconditions and Predicate Transformers

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


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
-----------------------------------	-----

$$\frac{C}{\emptyset} \quad | \quad \tau^C(\psi)$$

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

$$\frac{D}{\emptyset} \quad | \quad \tau^D(\psi)$$

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

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

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

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)

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

Rx1 r

Ry1 s

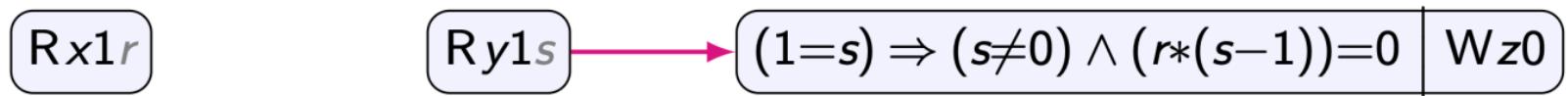
$(1 = s) \Rightarrow (s \neq 0) \wedge (r * (s - 1)) = 0$

Wz0

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

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

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

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

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

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

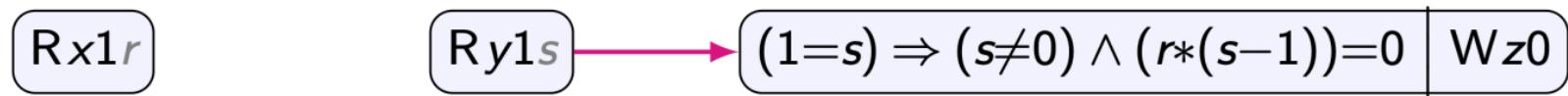
$$\frac{}{\begin{array}{c} C \\ \emptyset \\ \{Rx1r\} \end{array}} \quad \frac{}{\begin{array}{c} \tau^C(\psi) \\ \psi \\ (1=r) \Rightarrow \psi \end{array}}$$

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

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

Associative! (Right to Left)

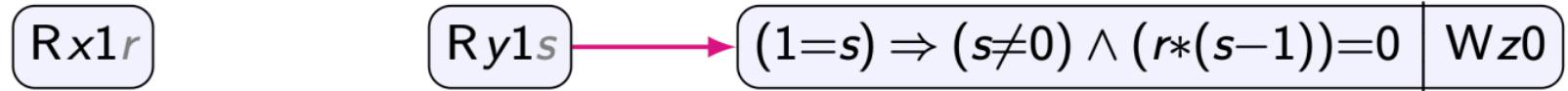
$r := x$ $(s := y \quad ; \quad \text{if } (s) \{ z := r * (s - 1) \})$



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

Associative! (Right to Left)

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

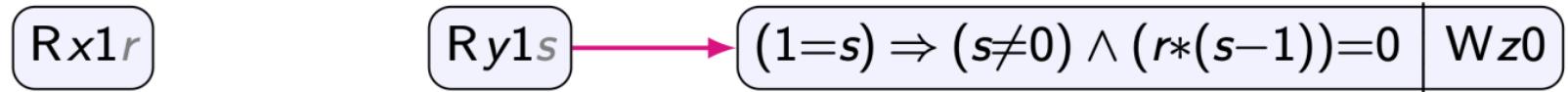


C	$\tau^C(\psi)$
\emptyset	ψ
$\{Rx1r\}$	$(1=r) \Rightarrow \psi$

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
$\{\text{Rx1}_r\}, \{\text{Rx1}_r, \text{Wy1}\}$	$(1=r) \Rightarrow \psi$
$\{\text{Ry1}_s\}, \{\text{Ry1}_s, \text{Wy1}\}$	$(1=s) \Rightarrow \psi$
$\{\text{Rx1}_r, \text{Ry1}_s\}, \{\text{Rx1}_r, \text{Ry1}_s, \text{Wy1}\}$	$(1=r) \Rightarrow (1=s) \Rightarrow \psi$

Associative! (Right to Left)

$r := x \quad ; \quad (s := y \quad ; \quad \text{if}(s) \{ z := r \cdot s \cdot \dots \})$

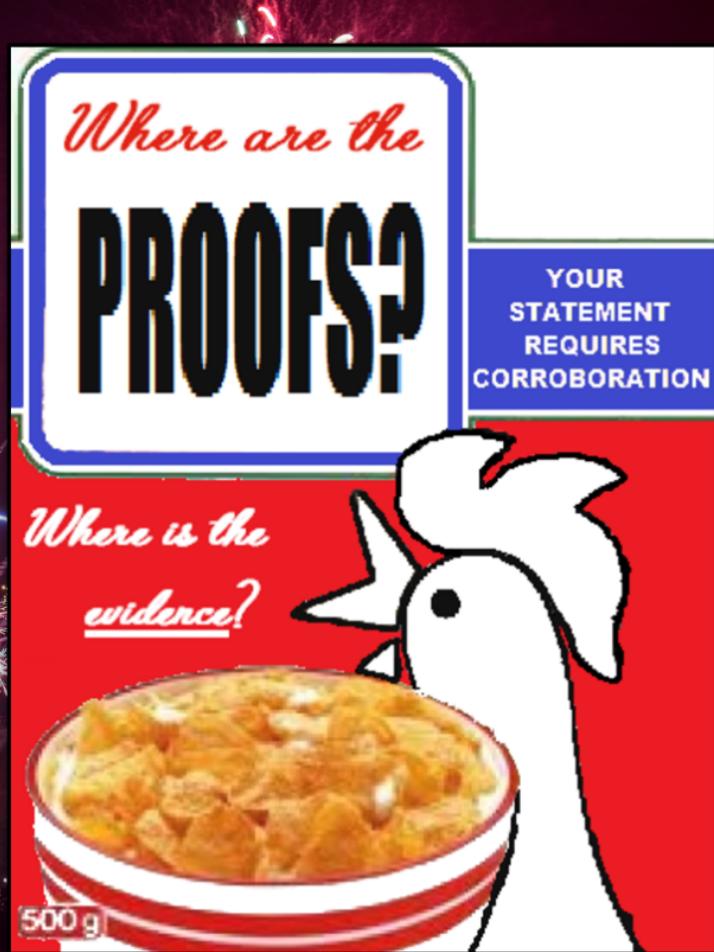
Rx1

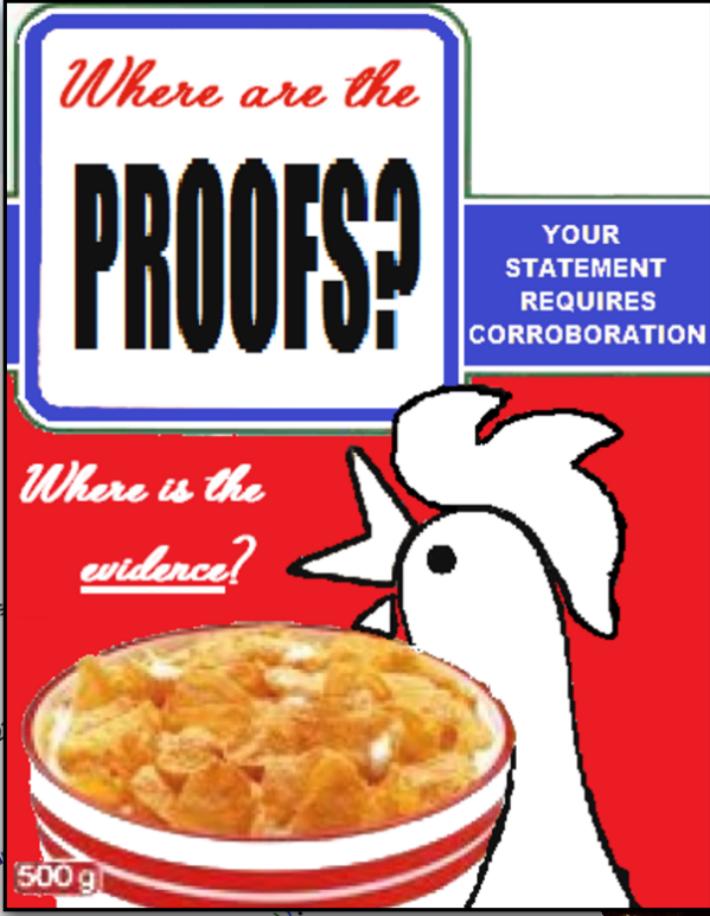
Wz0

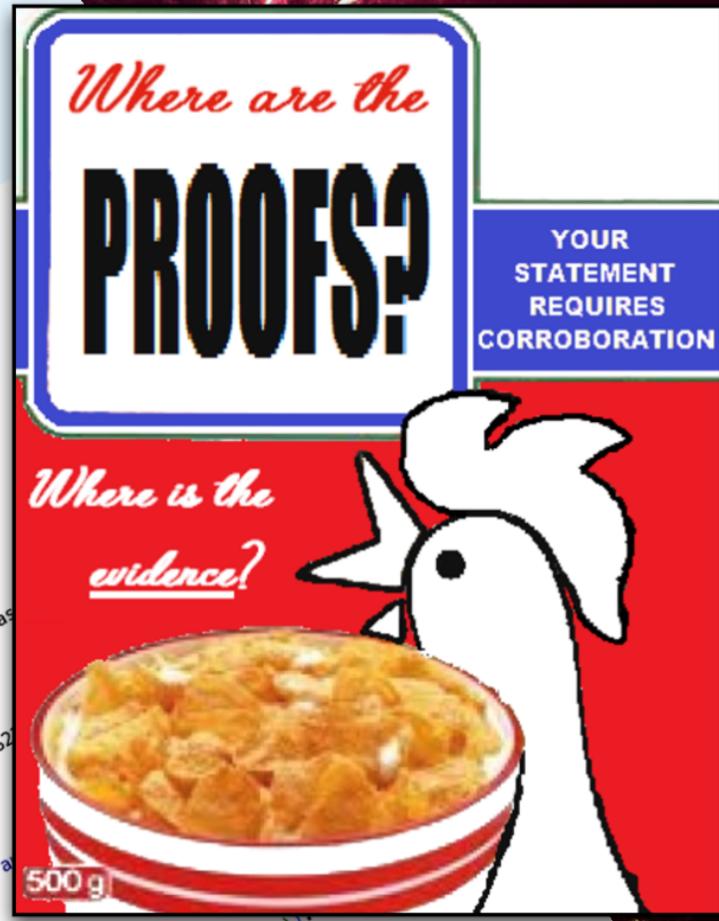
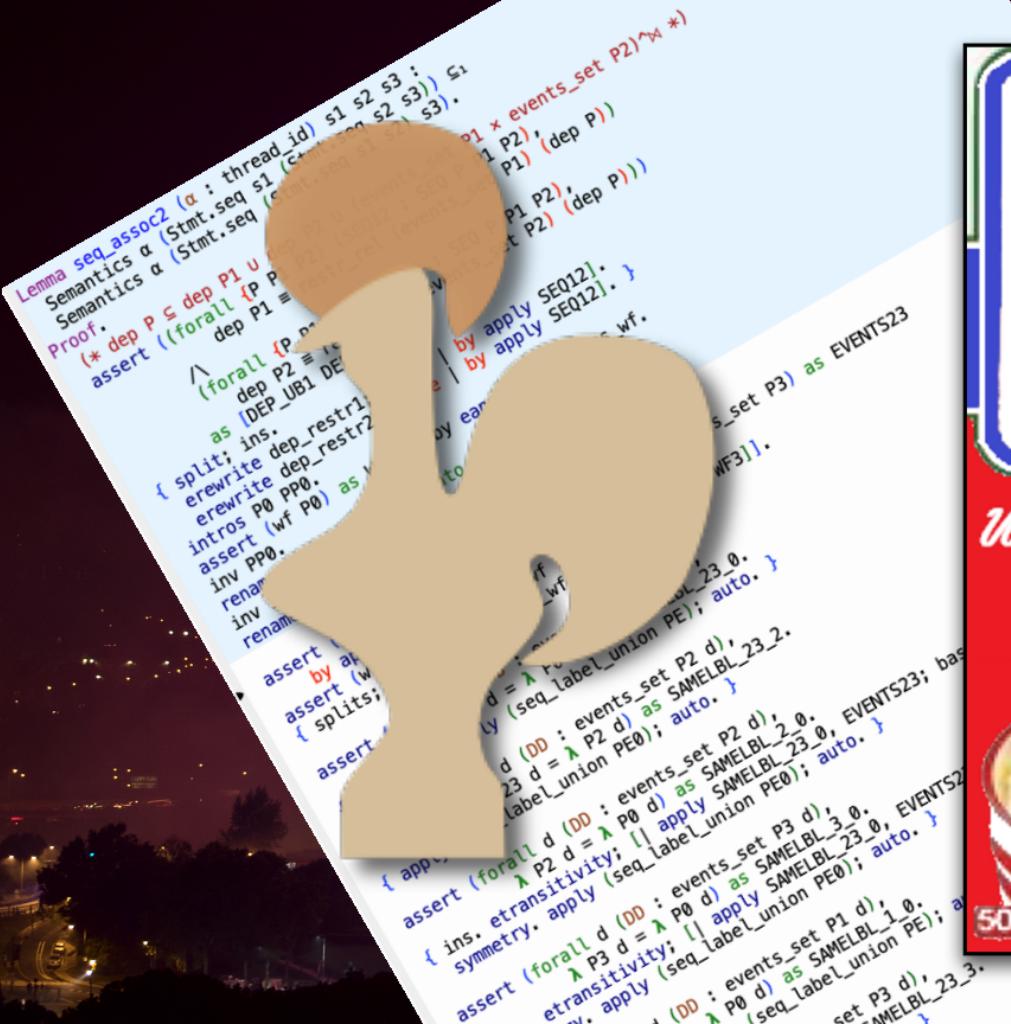
They're the same!

	$\tau^D(\psi)$
$\emptyset, \{Wy1\}$	ψ
$\{Rx1r\}, \{Rx1r, Wy1\}$	$(1=r) \Rightarrow \psi$
$\{Ry1s\}, \{Ry1s, Wy1\}$	$(1=s) \Rightarrow \psi$
$\{Rx1r, Ry1s\}, \{Rx1r, Ry1s, Wy1\}$	$(1=r) \Rightarrow (1=s) \Rightarrow \psi$









Real Hardware?



Logic=Local order

Pomset=Other order

```
assert !  
{ splits:  
  d_id) s1 s2 s3 :  
  G1(s1, s2, s3)) s1  
  o1 * events_set P2) ^x * )  
  1 P2) ' (dep P )  
  P1)  
{ app:  
  d (DD : events_set P2 )  
  label_union PE0); auto . }  
 assert (forall d (DD : events_set P2 )  
  P2 d = P0 d) as SAMELBL_23_0.  
 { ins, etransitivity;  
  [ apply SAMELBL_23_0, EVENTS23; base ]  
 symmetry. apply (seq_label_union PE0); auto . }  
 assert (forall d (DD : events_set P2 )  
  P3 d = P0 d) as SAMELBL_3_0.  
 { ins, etransitivity;  
  [ apply SAMELBL_3_0, EVENTS23; base ]  
 symmetry. apply (seq_label_union PE0); auto . }  
 assert (forall d (DD : events_set P1 )  
  P0 d) as SAMELBL_1_0.  
 { ins, etransitivity;  
  [ apply SAMELBL_1_0, EVENTS23; base ]  
 symmetry. apply (seq_label_union PE0); auto . }  
 assert (forall d (DD : events_set P2 )  
  P2 d = P0 d) as SAMELBL_23_3.  
 { ins, etransitivity;  
  [ apply SAMELBL_23_3, EVENTS23; base ]  
 symmetry. apply (seq_label_union PE0); auto . }
```



Logic=Local order
Pomset=Other order

MCA!

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 ✓

Copy Atomic Architectures

Non-MCA?

Let order include (in addition

Coherence (W_x)

Release/Acquisition/Fences (Wx)

SC A → SC B (W^{sc}_x → R^{sc}_y)

A: _____ or for: Relaxed ✓ Release ✓ Acqui-

Some work: Relaxed ✓ Release ✓ Acquire ✓

Armed Cats: Formal Conc
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

ANTON
LUC MARANG
We report on the process f

Example for handling mixed-size accesses applies to x86: We confirm this via
so show that our alternative formulations are applicable to any model

Computer systems organization → Architectures; Software properties; Theory of computation → Semantics
architecture, linux, mixed-size
Cats: 8

¹ Luc Maranget. 2021. Armed Cats: Article 8 (July 2021), 54 pages.

10

access
ACM Ref
la Alglar

Jade
Formal
<https://>

INTRODUCTION

1
A

INTRODUCTION
Arm has invested in forming the publication of a former in various ways, for example and necessity of that new structure cannot be understood. Arm has also developed in the catalogues.

al mode, by a mixed-size ensemble sounded on A developed and released language [19], a domain-executable manner. The C

in detail
formulation
writing

j.almer@arm.com, A.arm.com; luc.almer@gmail.com

Dependencies for C11!

Let order include:

Coherence

Release/Acquire

SC Acq

SC Rel

Acquire for: Relaxed ✓ Release ✓ Acquire ✓
some work: Relaxed ✓ Release ✓ Acquire ✓

Accesses ($VV \sim x \rightarrow R^{sc} y$)

Formal Concurrency Modelling at Arm
<https://doi.org/10.1145/3453401>

1 INTRODUCTION

Arm has invested in formal methods for consistency modelling in various ways, for example, by a formal model for memory access [18] and the publication of a formal model for the memory system [1]. The importance of mixed-size accesses [9]. The importance of the fact that without it Linux's lock structures cannot be used soundly [1]. All three models have been developed and released in the cat language [19], a domain-specific language for writing formal consistency models.

Written in idiomatic cat, as well as in formal languages such as Z and SPIN. The original paper [1] is available at <https://doi.org/10.1145/3453401>. The source code is available at <https://github.com/arm-labs/arm-concurrency-modelling>.

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

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

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

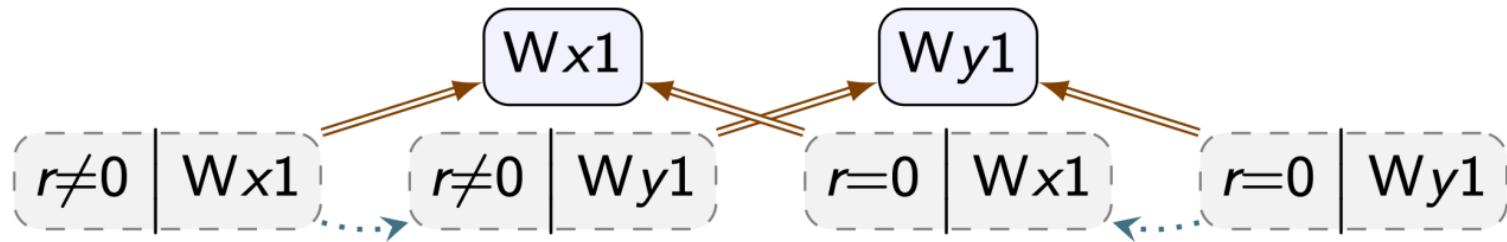
Merging and Synchronized-Before?

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



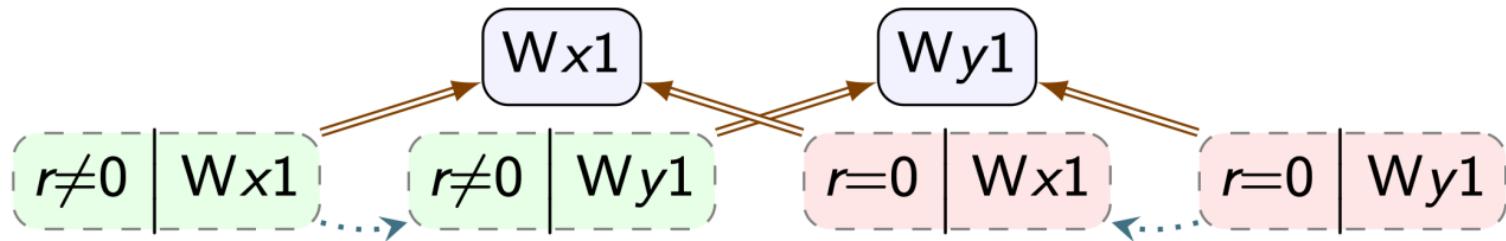
Merging and Synchronized-Before?

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



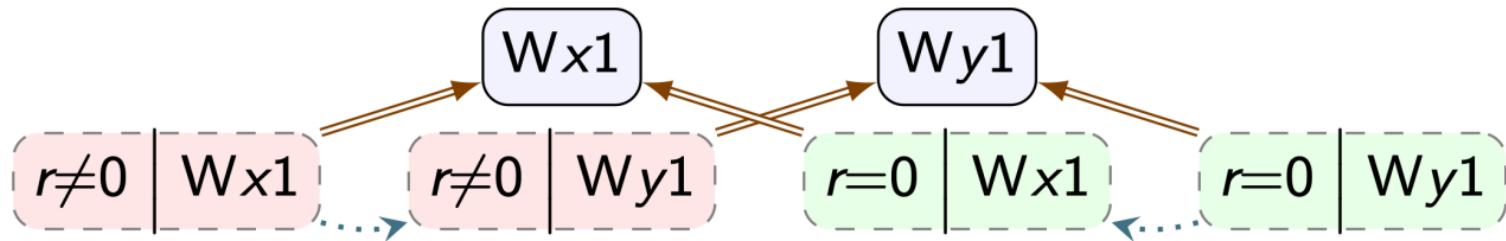
Merging and Synchronized-Before?

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



Merging and Synchronized-Before?

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



Merging and Synchronized-Before?

$\text{if}(r)$ -

```
$ cat data/tests/jctc/jctc1.lit
name=JCTC1
values={0,1}
comment "Should be allowed"
%%%
{
    r1 := x;
    if (r1 ≥ 0) {
        y := 1
    } else { skip }
} ||| {
    r2 := y;
    x := r2
}
%%%
```

$r \neq 0$

W

$x := 1\}$

$r = 0$

$W y 1$

```
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)
$
```

Merging and Synchronized-Before?

if(r) {

$r \neq 0$

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

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

$:= 1 \}$

$r = 0$ | Wy1

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

```
{  
    r1 =
```

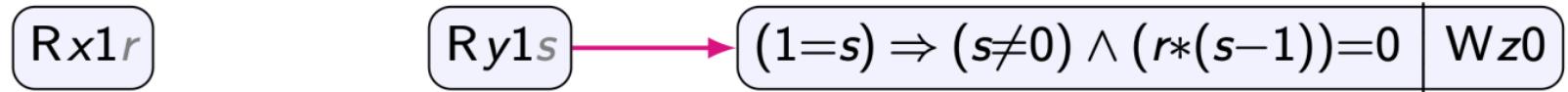
analysis could be moved early.

negative, allowing simplification of $r1 \geq 0$ to true, and allowing writ

```
$
```

Associative! (Right to Left)

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



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



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

© 2014 Pearson Education, Inc. All Rights Reserved. MCA

Concurrency is easy!

Tractable notion of...

Understandable by human beings

Sequentiality is hard!

More in the paper

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

Concurrency is easy!

Sequentiality is hard!

Concurrency is easy!

Sequentiality is hard!