On Stability in a Happens-Before Propagator for Concurrent Programs

Levente Bajczi, Csanád Telbisz, Dániel Szekeres, András Vörös

Dept. of AI and Systems Engineering, Budapest University of Technology and Economics

MŰEGYETEM 1782

BME FACULTY OF ELECTRICAL ENGINEERING AND INFORMATICS

ftsrg Critical Systems Research Group

Motivations

Motivations

- **Deagle** won CONCURRENCYSAFETY (SV-COMP) twice (2022-2023)

Motivations

- **Deagle** won CONCURRENCYSAFETY (SV-COMP) twice (2022-2023)

How?

- **Bounded Model Checking**
- **Consistency-Preserving Propagation [1]**
- **Decision Procedure for Happens-Before**

[1] Z. Sun, H. Fan, and F. He, “Consistency-preserving propagation for SMT solving of concurrent program verification,” Proc. ACM Program. Lang., vol. 6, no. OOPSLA2, Oct 2022.

Motivations

- **Deagle** won CONCURRENCYSAFETY (SV-COMP) twice (2022-2023)

How?

- **Bounded Model Checking**
- **Consistency-Preserving Propagation [1]**
- **Decision Procedure for Happens-Before**

- Logical next step…

[1] Z. Sun, H. Fan, and F. He, “Consistency-preserving propagation for SMT solving of concurrent program verification,” Proc. ACM Program. Lang., vol. 6, no. OOPSLA2, Oct 2022.

Motivations

- **Deagle** won ConcurrencySafety (SV-COMP) twice (2022-2023)

How?

- **Bounded Model Checking**
- **Consistency-Preserving Propagation [1]**
- **Decision Procedure for Happens-Before**

- Logical next step…

Let's re-implement it!

[1] Z. Sun, H. Fan, and F. He, "Consistency-preserving propagation for SMT solving of concurrent program verification," Proc. ACM Program. Lang., vol. 6, no. OOPSLA2, Oct 2022.

Contributions

Contributions

Theoretical analysis, counterexample to claims

Contributions

- Theoretical analysis, counterexample to claims
- Two proposals for fixing the algorithm

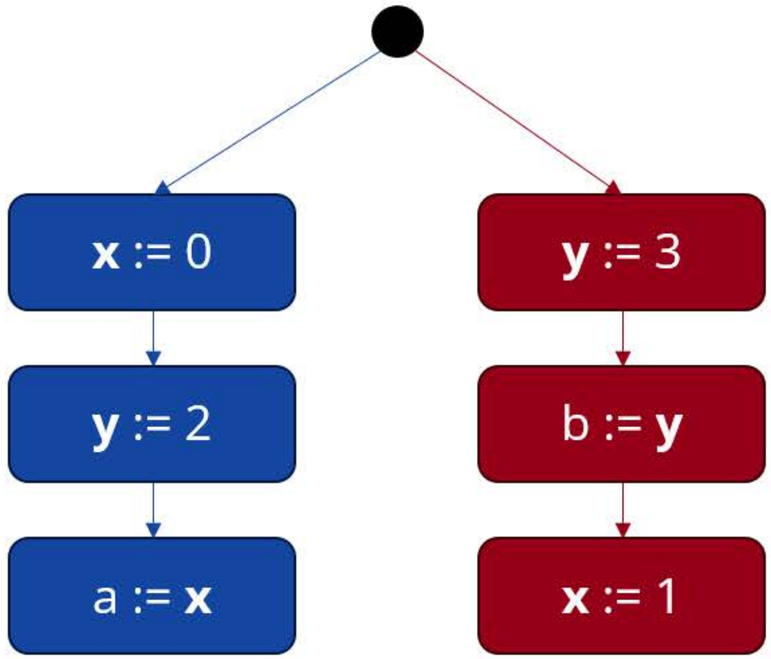
Contributions

- Theoretical analysis, counterexample to claims
- Two proposals for fixing the algorithm
- Prove original claim for both proposals

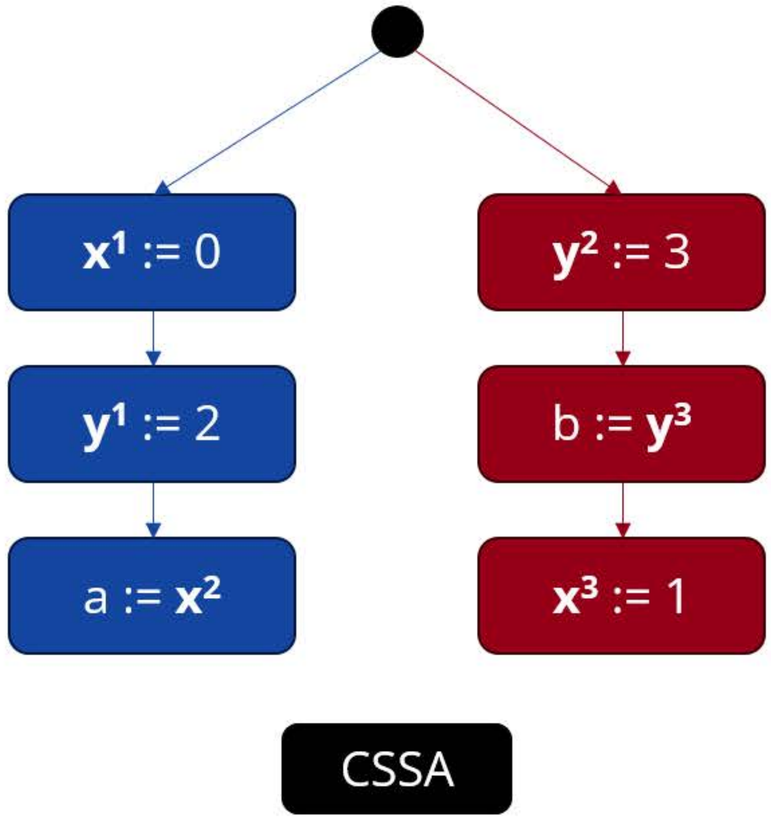
Contributions

- Theoretical analysis, counterexample to claims
- Two proposals for fixing the algorithm
- Prove original claim for both proposals
- Experimentally analyze performance impact

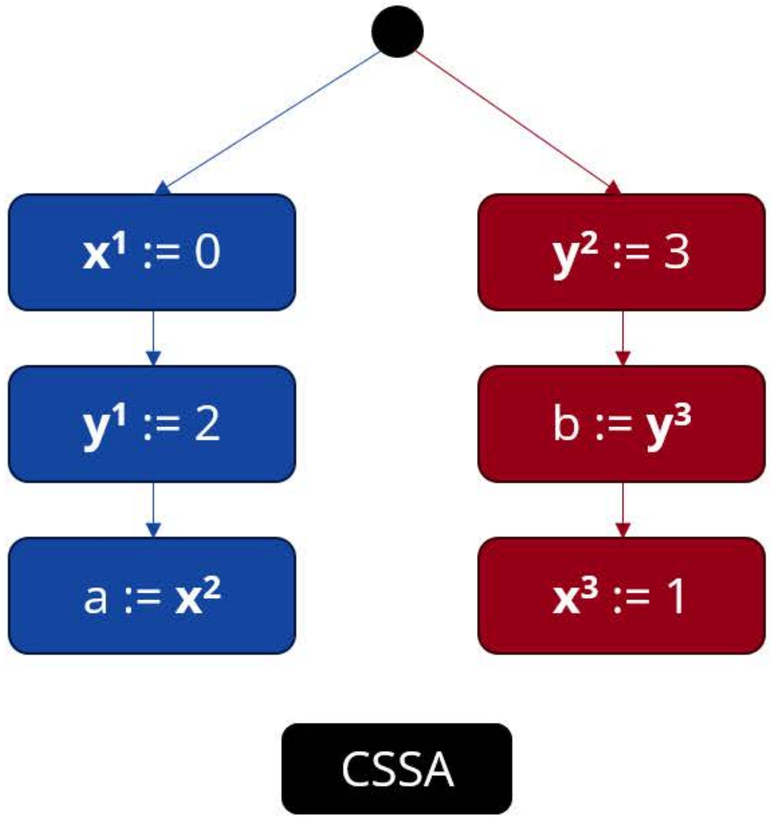
Verification with Happens-Before Orders



Verification with Happens-Before Orders

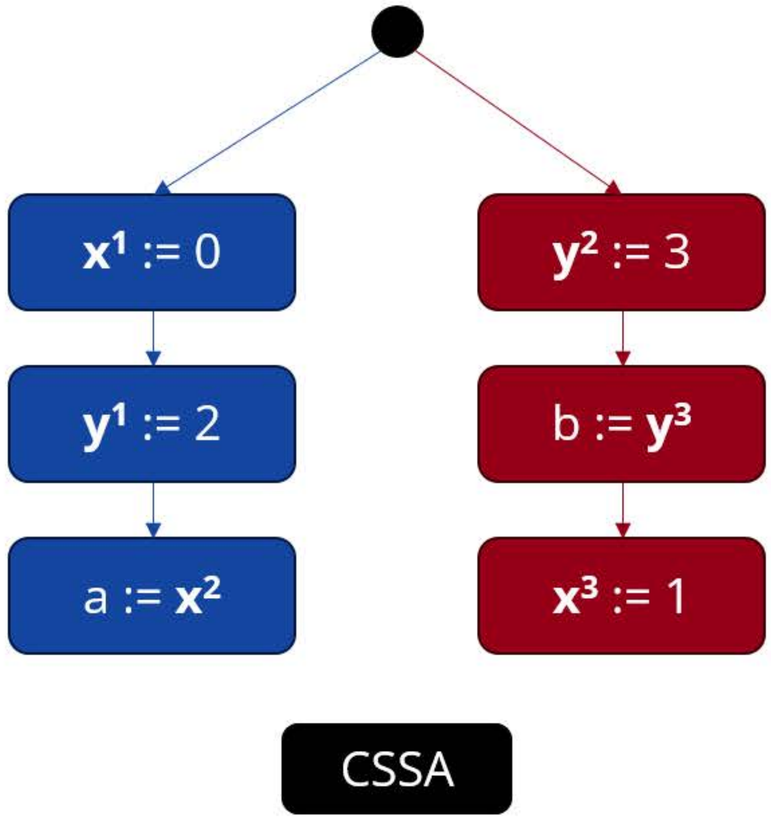


Verification with Happens-Before Orders



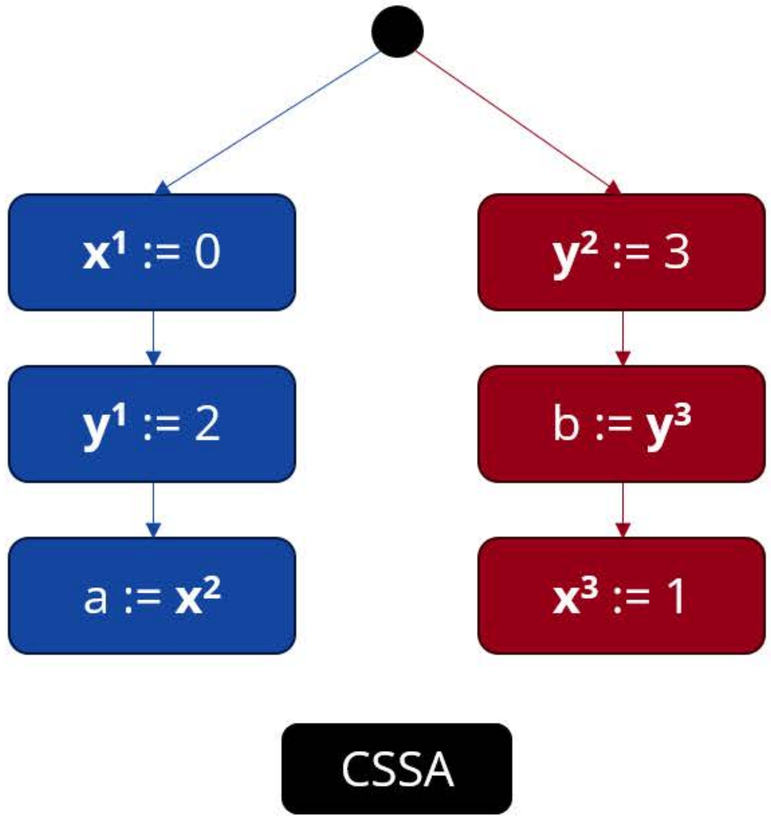
- ρ_{va}: write event values
  - $x^1==0$, $x^3==1$, $y^1==2$, $y^2==3$

Verification with Happens-Before Orders



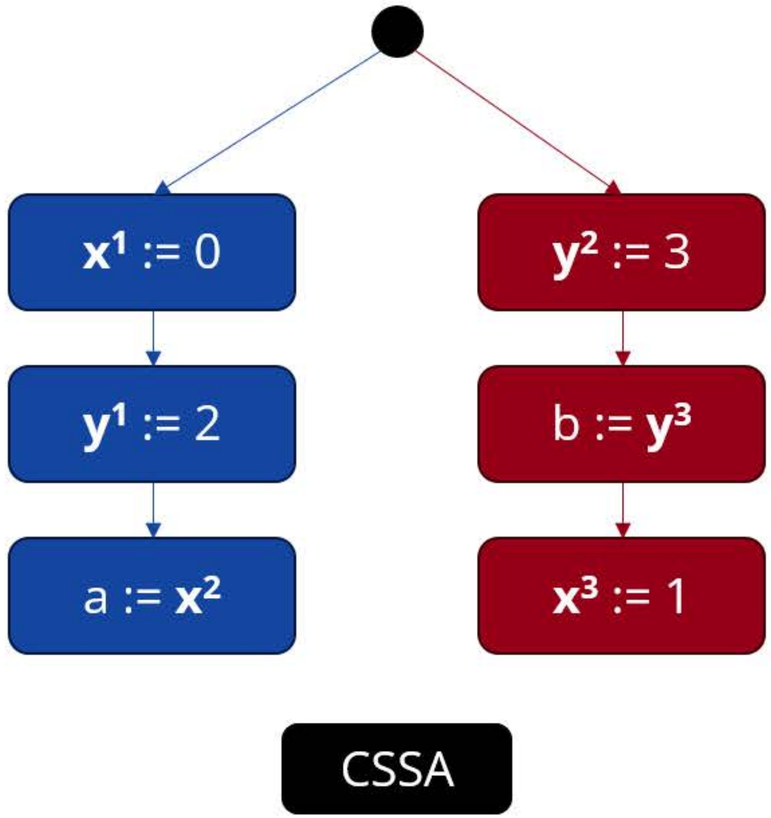
- ρ_{va}: write event values
  - $x^1==0,\ x^3==1,\ y^1==2,\ y^2==3$
- $\rho_{rf\text{-}val}$: read event values
  - $rf^x_{1,2} \Rightarrow x^1==x^2,\ rf^x_{3,2} \Rightarrow x^3==x^2$
    $rf^y_{1,3} \Rightarrow y^1==y^3,\ rf^y_{2,3} \Rightarrow y^2==y^3$

Verification with Happens-Before Orders



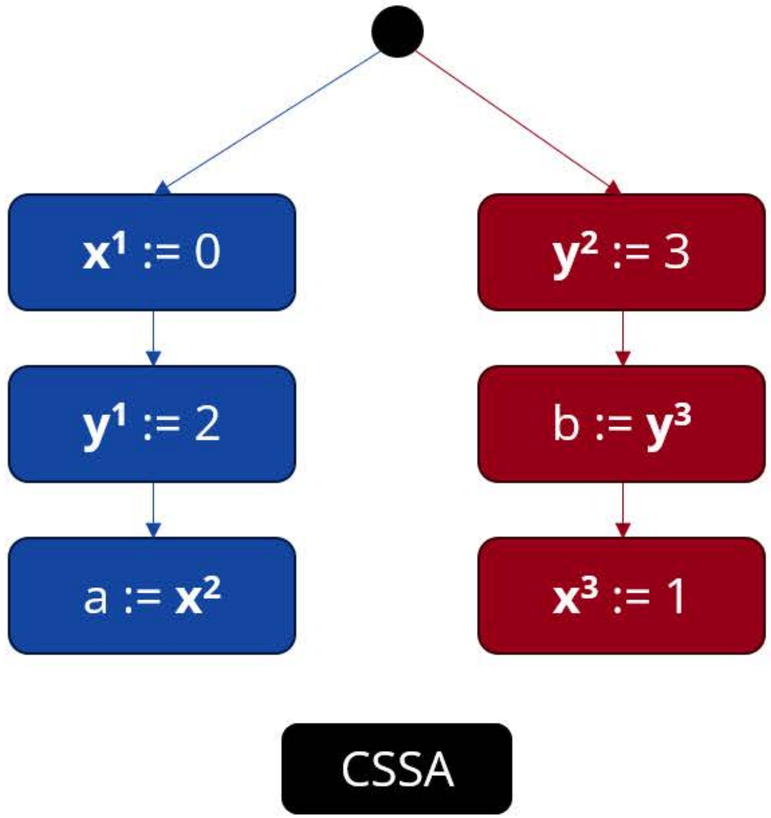
- ρ_{va}: write event values
  - $x^1==0,\ \ x^3==1,\ \ y^1==2,\ \ y^2==3$
- $\rho_{rf\text{-}val}$: read event values
  - $rf^x_{1,2} \Rightarrow x^1==x^2,\ \ rf^x_{3,2} \Rightarrow x^3==x^2$
    $rf^y_{1,3} \Rightarrow y^1==y^3,\ \ rf^y_{2,3} \Rightarrow y^2==y^3$
- $\rho_{rf\text{-}some}$: read event pairs
  - $rf^x_{1,2}\ xor\ rf^x_{3,2},\ \ rf^y_{1,3}\ xor\ rf^y_{2,3}$

Verification with Happens-Before Orders



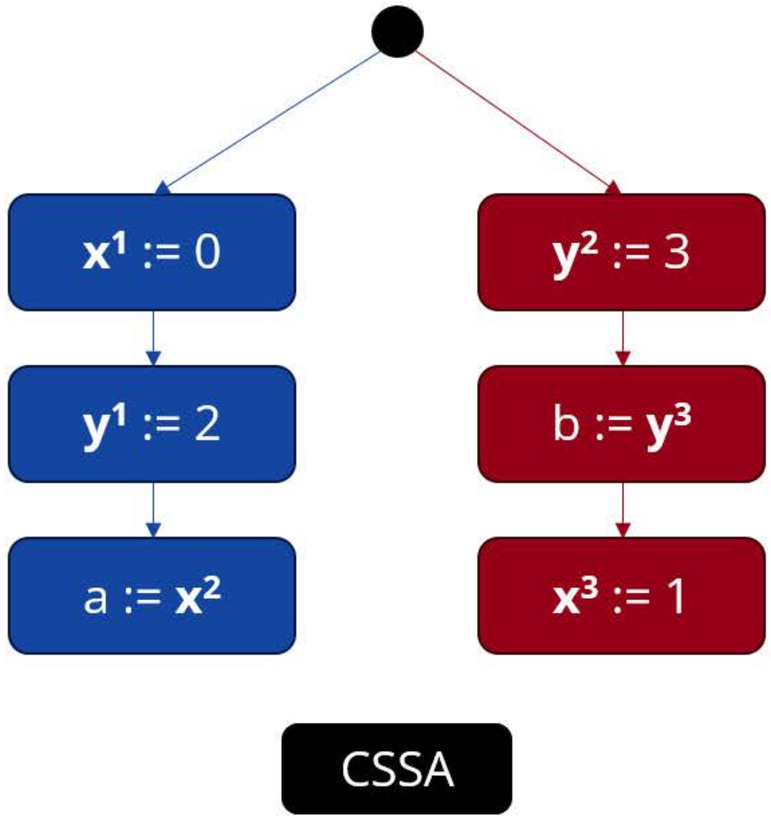
- ρ_{va}: write event values
  - $x^1==0,\ x^3==1,\ y^1==2,\ y^2==3$
- $\rho_{rf\text{-}val}$: read event values
  - $rf^x_{1,2} \Rightarrow x^1==x^2,\ rf^x_{3,2} \Rightarrow x^3==x^2$
    $rf^y_{1,3} \Rightarrow y^1==y^3,\ rf^y_{2,3} \Rightarrow y^2==y^3$
- $\rho_{rf\text{-}some}$: read event pairs
  - $rf^x_{1,2}\ xor\ rf^x_{3,2},\ rf^y_{1,3}\ xor\ rf^y_{2,3}$
- $\rho_{rf\text{-}ord}$: read-write ordering
  - $i \prec_{rf} j \iff rf_{ij}$
    $i \prec_{rf} j \Rightarrow i < j$

Verification with Happens-Before Orders



- ρ_{va}: write event values
  - $x^1==0,\ x^3==1,\ y^1==2,\ y^2==3$
- $\rho_{rf\text{-}val}$: read event values
  - $rf^x_{1,2} \Rightarrow x^1==x^2,\ rf^x_{3,2} \Rightarrow x^3==x^2$
    $rf^y_{1,3} \Rightarrow y^1==y^3,\ rf^y_{2,3} \Rightarrow y^2==y^3$
- $\rho_{rf\text{-}some}$: read event pairs
  - $rf^x_{1,2}\ xor\ rf^x_{3,2},\ rf^y_{1,3}\ xor\ rf^y_{2,3}$
- $\rho_{rf\text{-}ord}$: read-write ordering
  - $i \prec_{rf} j \iff rf_{ij}$
    $i \prec_{rf} j \Rightarrow i \prec j$
- ρ_{ppo}: same-thread order
  - $i \prec j$

Verification with Happens-Before Orders



- ρ_{va}: write event values
  - $x^1==0,\ x^3==1,\ y^1==2,\ y^2==3$
- $\rho_{rf\text{-}val}$: read event values
  - $rf^x_{1,2} \Rightarrow x^1==x^2,\ rf^x_{3,2} \Rightarrow x^3==x^2$
    $rf^y_{1,3} \Rightarrow y^1==y^3,\ rf^y_{2,3} \Rightarrow y^2==y^3$
- $\rho_{rf\text{-}some}$: read event pairs
  - $rf^x_{1,2}\ xor\ rf^x_{3,2},\ rf^y_{1,3}\ xor\ rf^y_{2,3}$
- $\rho_{rf\text{-}ord}$: read-write ordering
  - $i \prec_{rf} j \iff rf_{ij}$
    $i \prec_{rf} j \Rightarrow i \prec j$
- ρ_{ppo}: same-thread order
  - $i \prec j$

Happens-before

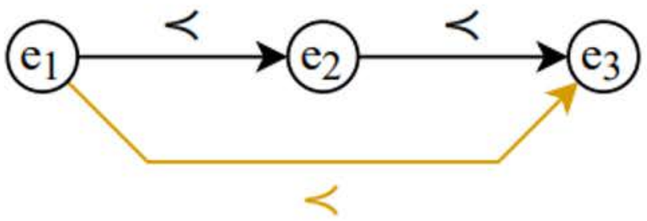
Verification with Happens-Before Orders

- $\rho_{\text{rf-ord}}$: read-write ordering
  - $i <_{rf} j \iff rf_{ij}$
    $i <_{rf} j \Rightarrow i < j$
- ρ_{ppo}: same-thread order
  - $i < j$

Verification with Happens-Before Orders

- $\rho_{\text{rf-ord}}$: read-write ordering
  - $i \prec_{rf} j \iff rf_{ij}$
    $i \prec_{rf} j \Rightarrow i \prec j$
- ρ_{ppo}: same-thread order
  - $i \prec j$

a) transitivity axiom:
$(e1 \prec e2) \land (e2 \prec e3) \Rightarrow (e1 \prec e3)$



(a) Transitivity

Verification with Happens-Before Orders

- $\rho_{rf\text{-}ord}$: read-write ordering
  - $i \prec_{rf} j \iff rf_{ij}$
    $i \prec_{rf} j \Rightarrow i \prec j$
- ρ_{ppo}: same-thread order
  - $i \prec j$

a) transitivity axiom:
$(e1 \prec e2) \land (e2 \prec e3) \Rightarrow (e1 \prec e3)$

b) write-serialization axiom:
$(w1 \prec_{rf} r) \land (w2 \prec r) \Rightarrow (w2 \prec w1)$

(a) Transitivity (b) Write-Serialization

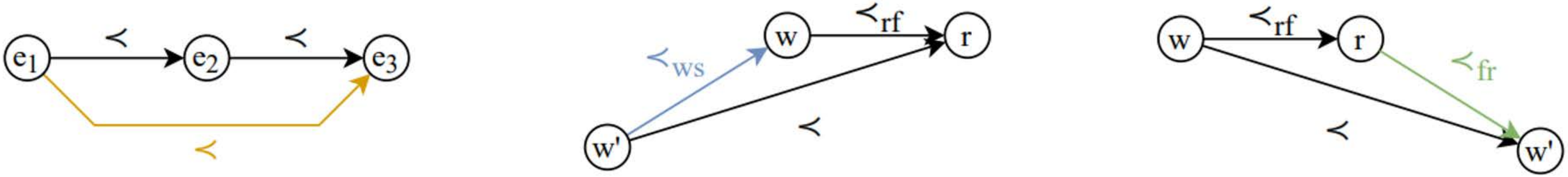
Verification with Happens-Before Orders

- $\rho_{\text{rf-ord}}$: read-write ordering
  - $i \prec_{rf} j \iff rf_{ij}$
    $i \prec_{rf} j \Rightarrow i \prec j$
- ρ_{ppo}: same-thread order
  - $i \prec j$

a) transitivity axiom:
$(e1 \prec e2) \land (e2 \prec e3) \Rightarrow (e1 \prec e3)$

b) write-serialization axiom:
$(w1 \prec_{rf} r) \land (w2 \prec r) \Rightarrow (w2 \prec w1)$

c) from-read axiom:
$(w1 \prec_{rf} r) \land (w1 \prec w2) \Rightarrow (r \prec w2)$



(a) Transitivity (b) Write-Serialization (c) From-Read

Verification with Happens-Before Orders

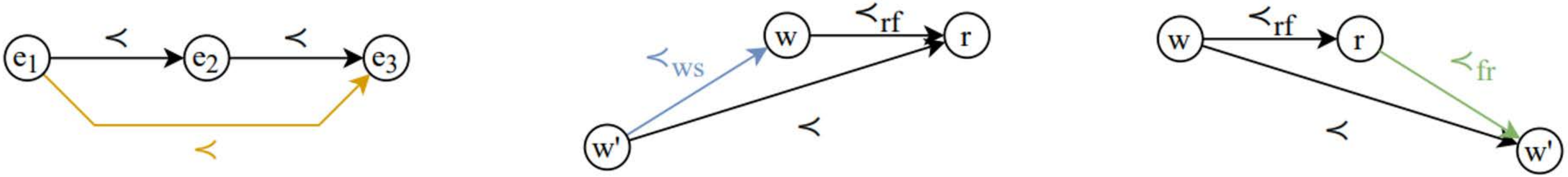
- $\rho_{\text{rf-ord}}$: read-write ordering
  - $i \prec_{rf} j \iff rf_{ij}$
    $i \prec_{rf} j \Rightarrow i \prec j$
- ρ_{ppo}: same-thread order
  - $i \prec j$

a) transitivity axiom:
$(e1 \prec e2) \land (e2 \prec e3) \Rightarrow (e1 \prec e3)$

b) write-serialization axiom:
$(w1 \prec_{rf} r) \land (w2 \prec r) \Rightarrow (w2 \prec w1)$

c) from-read axiom:
$(w1 \prec_{rf} r) \land (w1 \prec w2) \Rightarrow (r \prec w2)$

$\prec$ should be acylic



(a) Transitivity (b) Write-Serialization (c) From-Read

Consistency-Preserving Propagation

Consistency-Preserving Propagation

- Encode SMT formulae
- On branching: PROPAGATE()

Consistency-Preserving Propagation

- Encode SMT formulae
- On branching: PROPAGATE()

Algorithm 1 Original Theory Propagation Algorithm [5], with typos fixed

```
Input: l: Positive literal
proc PROPAGATE(l)
    if l is rf_ij then
        w_i, r_j := write and read of rf_ij
        ≺_rf ← ≺_rf ∪ {(w_i, r_j)}
        ≺ ← ≺ ∪ {(w_i, r_j)}
        TRANSITIVITY(w_i, r_j)
        foreach w_k s.t. grd_k do
            if (w_i, w_k, r_j) same-addr. then
                DERIVE(w_i, w_k, r_j)
            end if
        end foreach
    else if l is grd_k then
        w_k := write of grd_k
        foreach (w_i, r_i) ∈ ≺_rf do
            if (w_i, w_k, r_j) same-addr. then
                DERIVE(w_i, w_k, r_j)
            end if
        end foreach
    end if
end proc
```

```
Input: w_i, w_k, r_j: w_i ≺_rf r_j, grd_k
proc DERIVE(w_i, w_k, r_j)
    if w_k ≺ r_j then
        ≺ ← ≺ ∪ {(w_k, w_i)}
        TRANSITIVITY(w_k, w_i)
    else if w_i ≺ w_k then
        ≺ ← ≺ ∪ {(r_j, w_k)}
        TRANSITIVITY(r_j, w_k)
    end if
end proc
```

```
Input: e_1, e_2: e_1 ≺ e_2
proc TRANSITIVITY(e_1, e_2)
    foreach (e_2, e_3) ∈ ≺ do
        if (e_1, e_3) ∉ ≺ then
            ≺ ← ≺ ∪ {(e_1, e_3)}
            TRANSITIVITY(e_1, e_3)
        end if
    end foreach
    foreach (e_0, e_1) ∈ ≺ do
        if (e_0, e_2) ∉ ≺ then
            ≺ ← ≺ ∪ {(e_0, e_2)}
            TRANSITIVITY(e_0, e_2)
        end if
    end foreach
end proc
```

Consistency-Preserving Propagation

- Encode SMT formulae
- On branching: PROPAGATE()

Fewer backtracks (none due to $\prec$)

Algorithm 1 Original Theory Propagation Algorithm [5], with typos fixed

```
Input: l: Positive literal
proc PROPAGATE(l)
    if l is rf_ij then
        w_i, r_j := write and read of rf_ij
        ≺_rf ← ≺_rf ∪ {(w_i, r_j)}
        ≺ ← ≺ ∪ {(w_i, r_j)}
        TRANSITIVITY(w_i, r_j)
        foreach w_k s.t. grd_k do
            if (w_i, w_k, r_j) same-addr. then
                DERIVE(w_i, w_k, r_j)
            end if
        end foreach
    else if l is grd_k then
        w_k := write of grd_k
        foreach (w_i, r_i) ∈ ≺_rf do
            if (w_i, w_k, r_j) same-addr. then
                DERIVE(w_i, w_k, r_j)
            end if
        end foreach
    end if
end proc
```

```
Input: w_i, w_k, r_j: w_i ≺_rf r_j, grd_k
proc DERIVE(w_i, w_k, r_j)
    if w_k ≺ r_j then
        ≺ ← ≺ ∪ {(w_k, w_i)}
        TRANSITIVITY(w_k, w_i)
    else if w_i ≺ w_k then
        ≺ ← ≺ ∪ {(r_j, w_k)}
        TRANSITIVITY(r_j, w_k)
    end if
end proc
```

```
Input: e_1, e_2: e_1 ≺ e_2
proc TRANSITIVITY(e_1, e_2)
    foreach (e_2, e_3) ∈ ≺ do
        if (e_1, e_3) ∉ ≺ then
            ≺ ← ≺ ∪ {(e_1, e_3)}
            TRANSITIVITY(e_1, e_3)
        end if
    end foreach
    foreach (e_0, e_1) ∈ ≺ do
        if (e_0, e_2) ∉ ≺ then
            ≺ ← ≺ ∪ {(e_0, e_2)}
            TRANSITIVITY(e_0, e_2)
        end if
    end foreach
end proc
```

Consistency-Preserving Propagation

- Encode SMT formulae
- On branching: PROPAGATE()

Fewer backtracks (none due to $\prec$)

Claim: after PROPAGATE(), $\prec$ is *stable*

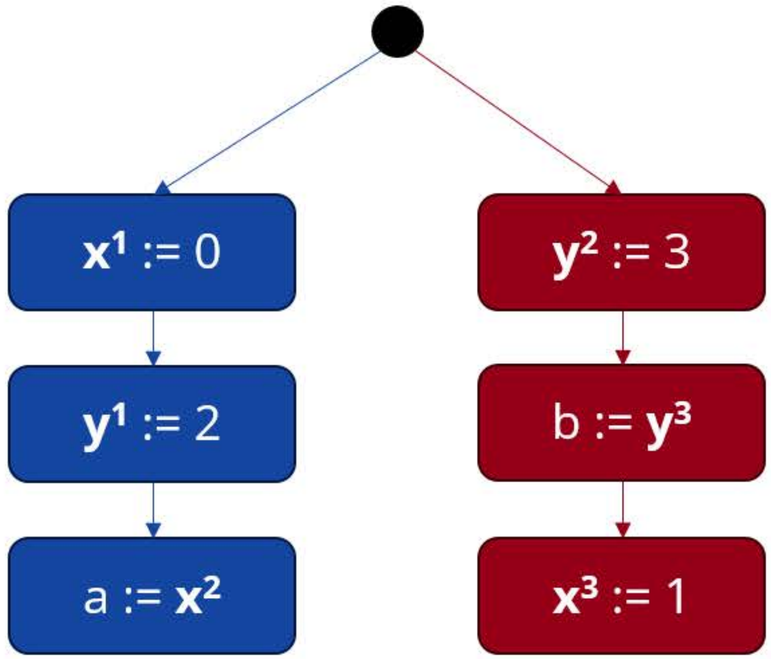
Algorithm 1 Original Theory Propagation Algorithm [5], with typos fixed

Input: l: Positive literal
proc PROPAGATE(l)
  if l is rf_{ij} **then**
    $w_i, r_j :=$ write and read of rf_{ij}
    $\prec_{rf} \leftarrow \prec_{rf} \cup \{(w_i, r_j)\}$
    $\prec \leftarrow \prec \cup \{(w_i, r_j)\}$
    TRANSITIVITY(w_i, r_j)
    foreach w_k *s.t.* grd_k **do**
      if (w_i, w_k, r_j) same-addr. **then**
        DERIVE(w_i, w_k, r_j)
      end if
    end foreach
  else if l is grd_k **then**
    $w_k :=$ write of grd_k
    foreach $(w_i, r_i) \in \prec_{rf}$ **do**
      if (w_i, w_k, r_j) same-addr. **then**
        DERIVE(w_i, w_k, r_j)
      end if
    end foreach
  end if
end proc

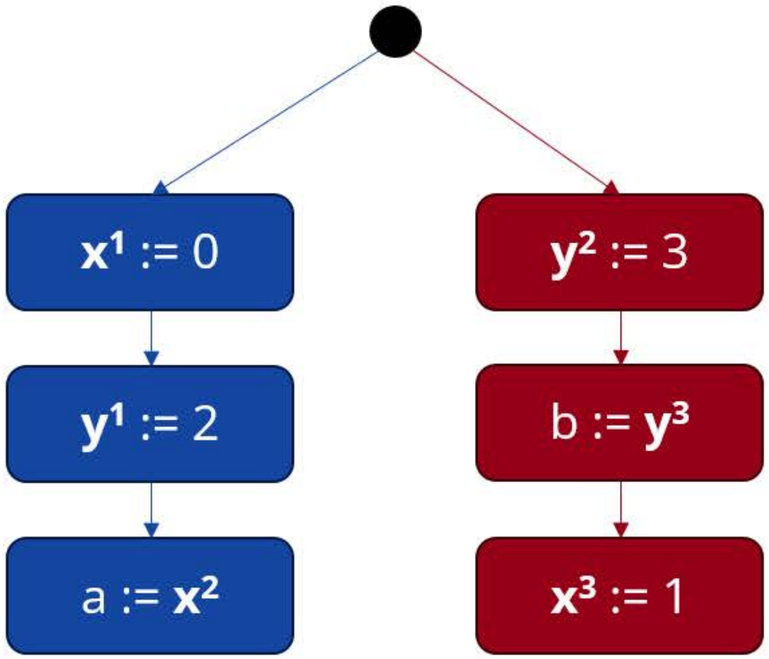
Input: w_i, w_k, r_j: $w_i \prec_{rf} r_j, grd_k$
proc DERIVE(w_i, w_k, r_j)
  if $w_k \prec r_j$ **then**
    $\prec \leftarrow \prec \cup \{(w_k, w_i)\}$
    TRANSITIVITY(w_k, w_i)
  else if $w_i \prec w_k$ **then**
    $\prec \leftarrow \prec \cup \{(r_j, w_k)\}$
    TRANSITIVITY(r_j, w_k)
  end if
end proc

Input: e_1, e_2: $e_1 \prec e_2$
proc TRANSITIVITY(e_1, e_2)
  foreach $(e_2, e_3) \in \prec$ **do**
    if $(e_1, e_3) \notin \prec$ **then**
      $\prec \leftarrow \prec \cup \{(e_1, e_3)\}$
      TRANSITIVITY(e_1, e_3)
    end if
  end foreach
  foreach $(e_0, e_1) \in \prec$ **do**
    if $(e_0, e_2) \notin \prec$ **then**
      $\prec \leftarrow \prec \cup \{(e_0, e_2)\}$
      TRANSITIVITY(e_0, e_2)
    end if
  end foreach
end proc

Consistency-Preserving Propagation



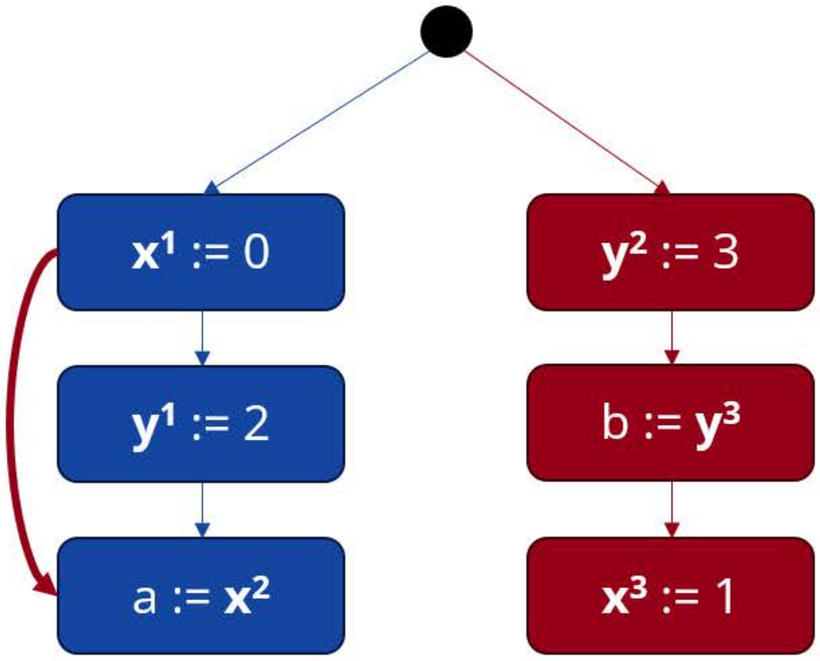
Consistency-Preserving Propagation



propagate($rf^x_{1,2}$)

$x_1 \prec x_2$ $x_1 \prec_{rf} x_2$

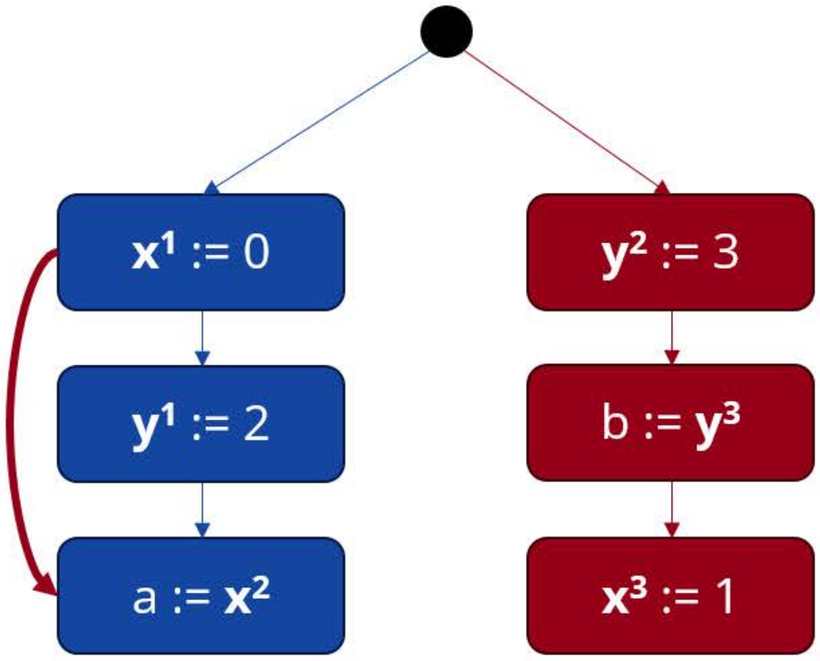
Consistency-Preserving Propagation



propagate($rf^x_{1,2}$)

$x_1 \prec x_2$ $x_1 \prec_{rf} x_2$

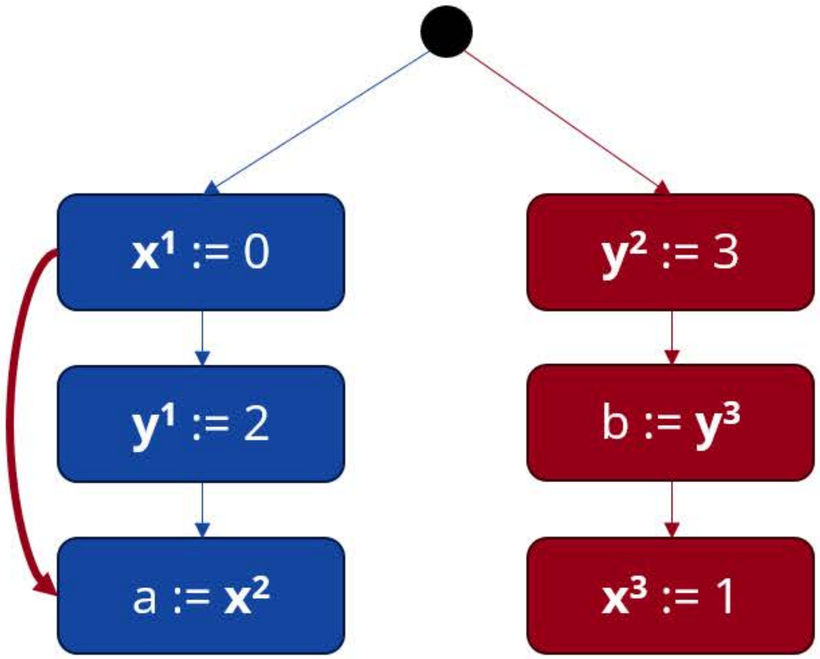
Consistency-Preserving Propagation



propagate($rf^x_{1,2}$)
transitivity(x_1, x_2)

$x_1 \prec x_2$ $x_1 \prec_{rf} x_2$

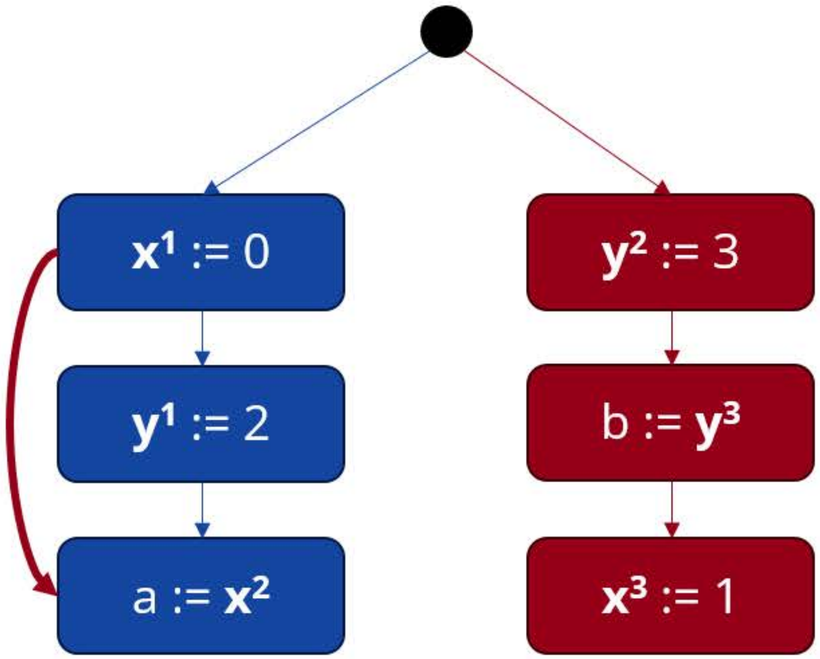
Consistency-Preserving Propagation



propagate($rf^x_{1,2}$)
transitivity(x_1, x_2)
derive(x_1, x_3, x_2)

$x_1 \prec x_2$ $x_1 \prec_{rf} x_2$

Consistency-Preserving Propagation



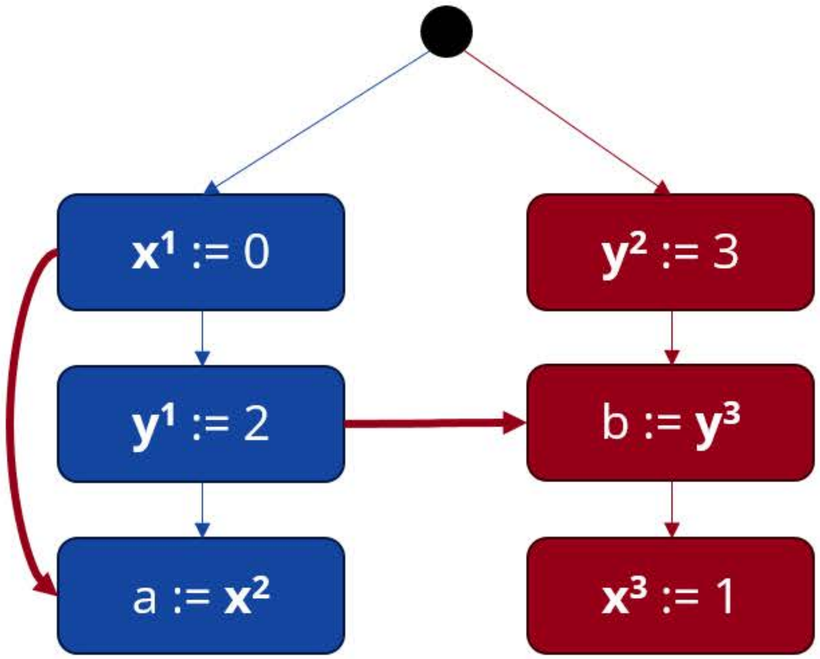
propagate($rf^x_{1,2}$) $\quad x_1 \prec x_2 \quad x_1 \prec_{rf} x_2$

transitivity(x_1, x_2)

derive(x_1, x_3, x_2)

propagate($rf^y_{1,3}$) $\quad y_1 \prec y_3 \quad y_1 \prec_{rf} y_3$

Consistency-Preserving Propagation



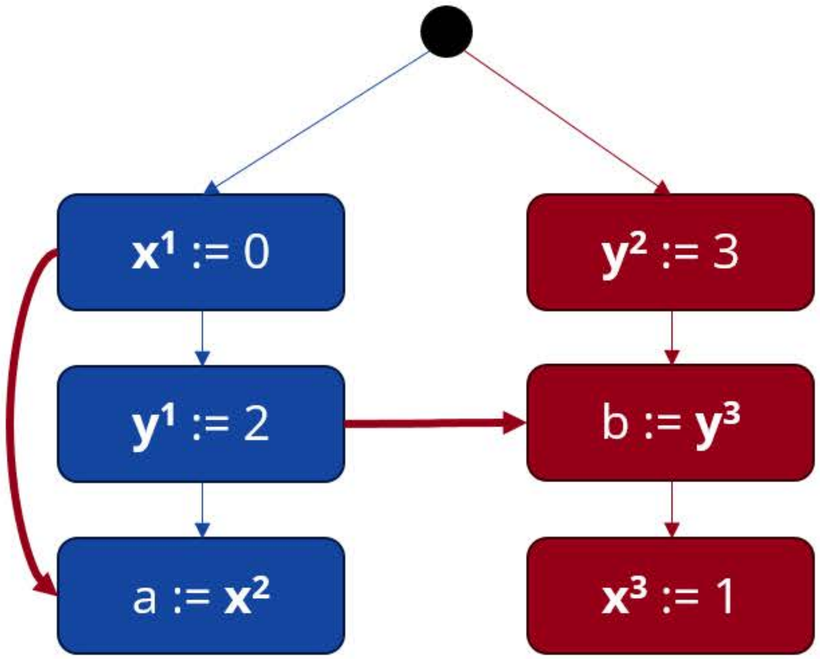
propagate($rf^x_{1,2}$) $\quad x_1 \prec x_2 \quad x_1 \prec_{rf} x_2$

transitivity(x_1, x_2)

derive(x_1, x_3, x_2)

propagate($rf^y_{1,3}$) $\quad y_1 \prec y_3 \quad y_1 \prec_{rf} y_3$

Consistency-Preserving Propagation



propagate($rf^x_{1,2}$) $\quad$ $x_1 \prec x_2$ $\quad$ $x_1 \prec_{rf} x_2$

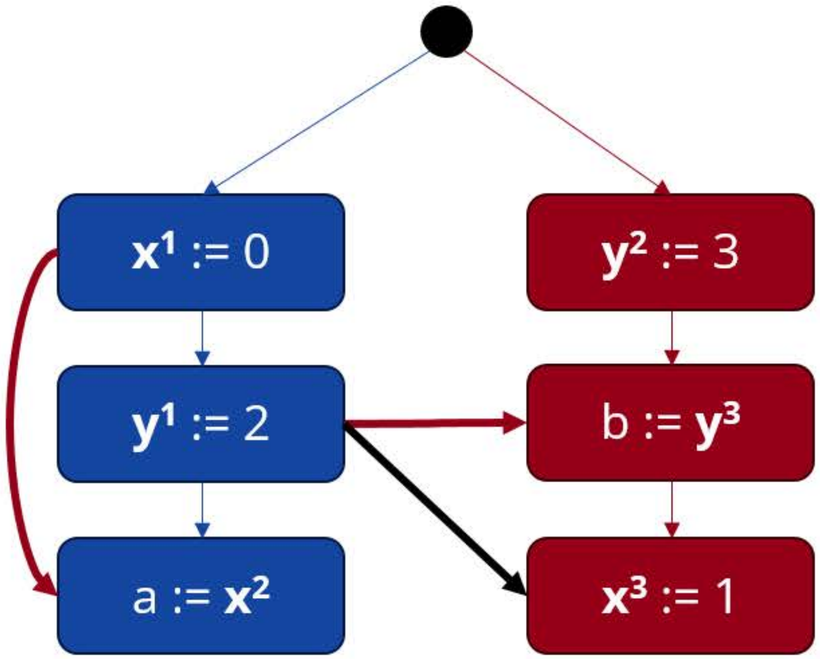
transitivity(x_1, x_2)

derive(x_1, x_3, x_2)

propagate($rf^y_{1,3}$) $\quad$ $y_1 \prec y_3$ $\quad$ $y_1 \prec_{rf} y_3$

transitivity(y_1, y_3) $\quad$ $y_1 \prec x_3$

Consistency-Preserving Propagation



propagate($rf^x_{1,2}$) $\quad$ $x_1 \prec x_2$ $\quad$ $x_1 \prec_{rf} x_2$

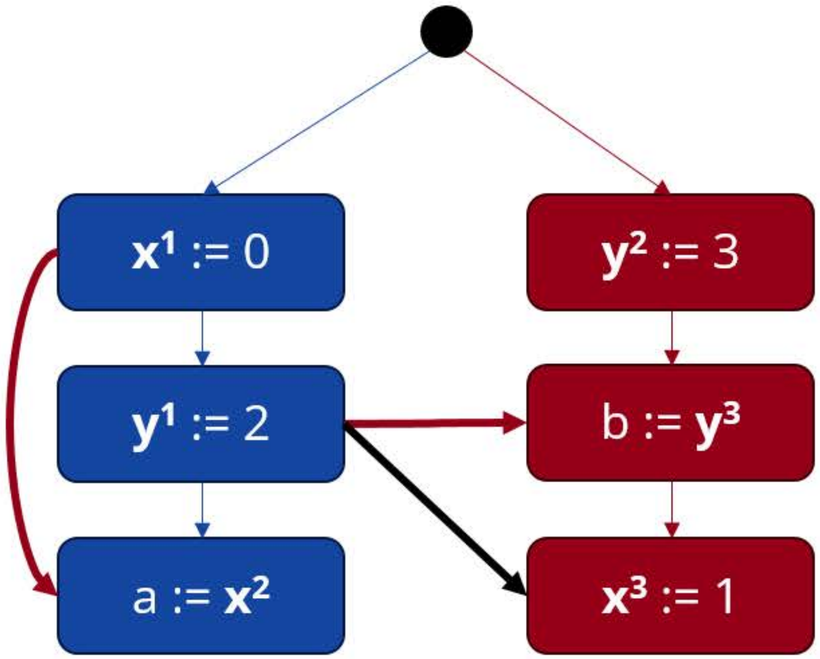
transitivity(x_1, x_2)

derive(x_1, x_3, x_2)

propagate($rf^y_{1,3}$) $\quad$ $y_1 \prec y_3$ $\quad$ $y_1 \prec_{rf} y_3$

transitivity(y_1, y_3) $\quad$ $y_1 \prec x_3$

Consistency-Preserving Propagation



propagate($rf^x_{1,2}$) $\quad$ $x_1 \prec x_2$ $\quad$ $x_1 \prec_{rf} x_2$

transitivity(x_1, x_2)

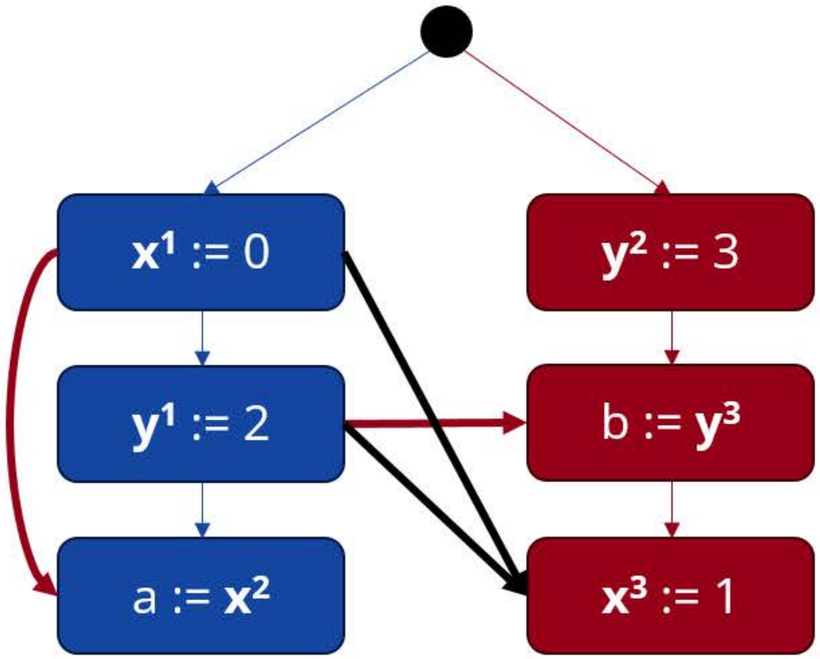
derive(x_1, x_3, x_2)

propagate($rf^y_{1,3}$) $\quad$ $y_1 \prec y_3$ $\quad$ $y_1 \prec_{rf} y_3$

transitivity(y_1, y_3) $\quad$ $y_1 \prec x_3$

transitivity(y_1, x_3) $\quad$ $x_1 \prec x_3$

Consistency-Preserving Propagation



propagate($rf^x_{1,2}$) $\quad$ $x_1 \prec x_2$ $\quad$ $x_1 \prec_{rf} x_2$

transitivity(x_1, x_2)

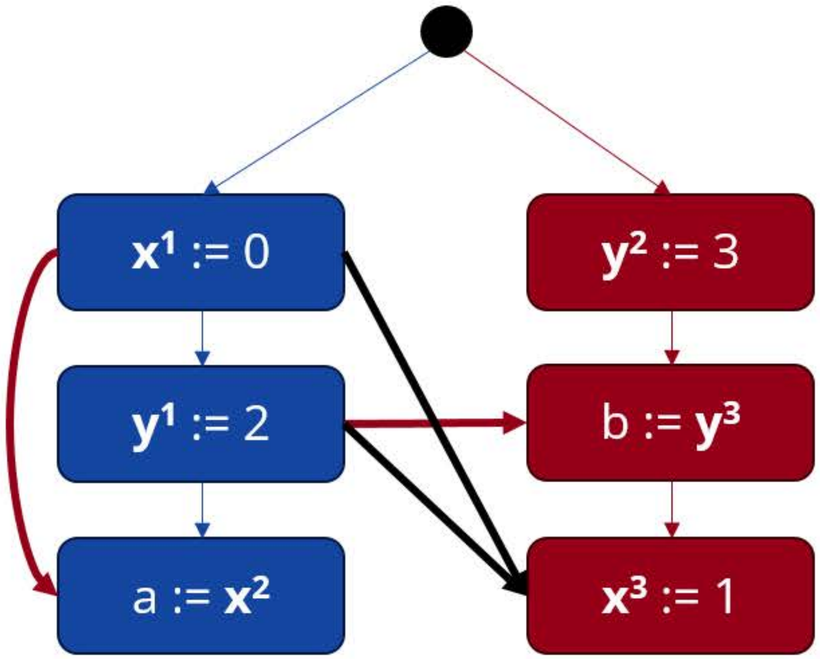
derive(x_1, x_3, x_2)

propagate($rf^y_{1,3}$) $\quad$ $y_1 \prec y_3$ $\quad$ $y_1 \prec_{rf} y_3$

transitivity(y_1, y_3) $\quad$ $y_1 \prec x_3$

transitivity(y_1, x_3) $\quad$ $x_1 \prec x_3$

Consistency-Preserving Propagation



propagate($rf^x_{1,2}$) $\quad$ $x_1 \prec x_2$ $\quad$ $x_1 \prec_{rf} x_2$

transitivity(x_1, x_2)

derive(x_1, x_3, x_2)

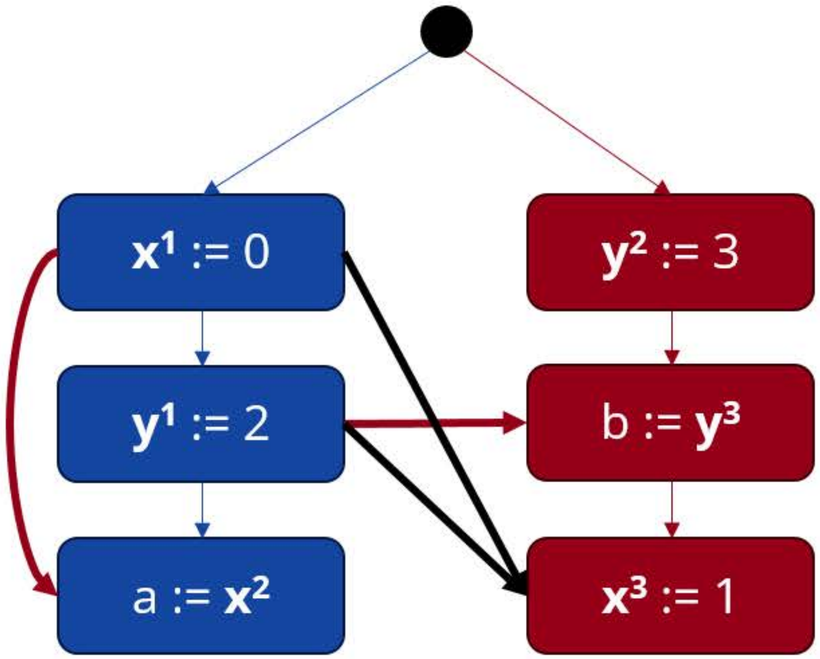
propagate($rf^y_{1,3}$) $\quad$ $y_1 \prec y_3$ $\quad$ $y_1 \prec_{rf} y_3$

transitivity(y_1, y_3) $\quad$ $y_1 \prec x_3$

transitivity(y_1, x_3) $\quad$ $x_1 \prec x_3$

transitivity(x_1, x_3)

Consistency-Preserving Propagation



propagate($rf^x_{1,2}$) $\quad x_1 \prec x_2 \quad x_1 \prec_{rf} x_2$

transitivity(x_1, x_2)

derive(x_1, x_3, x_2)

propagate($rf^y_{1,3}$) $\quad y_1 \prec y_3 \quad y_1 \prec_{rf} y_3$

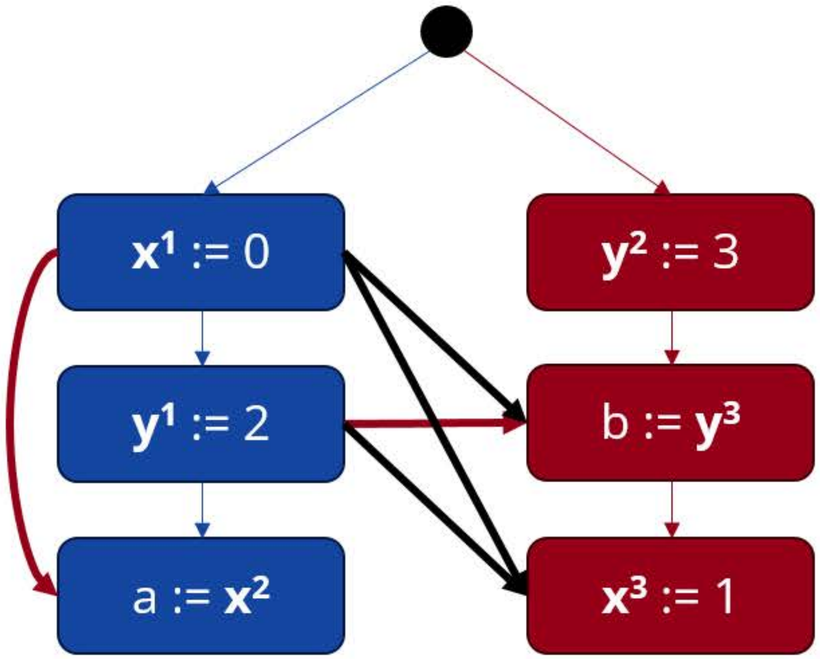
transitivity(y_1, y_3) $\quad y_1 \prec x_3$

transitivity(y_1, x_3) $\quad x_1 \prec x_3$

transitivity(x_1, x_3)

$x_1 \prec y_3$

Consistency-Preserving Propagation



propagate($rf^x_{1,2}$) $\quad x_1 \prec x_2 \quad x_1 \prec_{rf} x_2$

transitivity(x_1, x_2)

derive(x_1, x_3, x_2)

propagate($rf^y_{1,3}$) $\quad y_1 \prec y_3 \quad y_1 \prec_{rf} y_3$

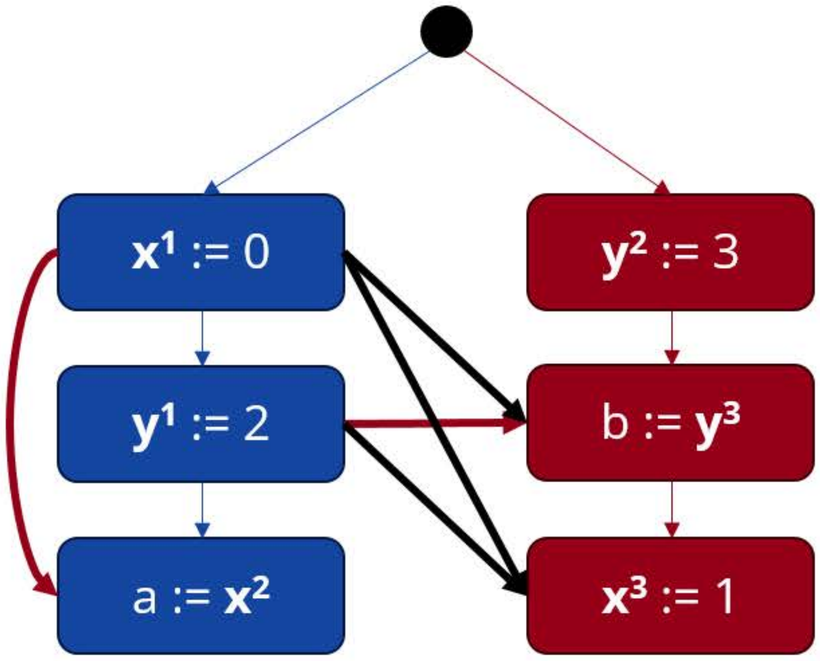
transitivity(y_1, y_3) $\quad y_1 \prec x_3$

transitivity(y_1, x_3) $\quad x_1 \prec x_3$

transitivity(x_1, x_3)

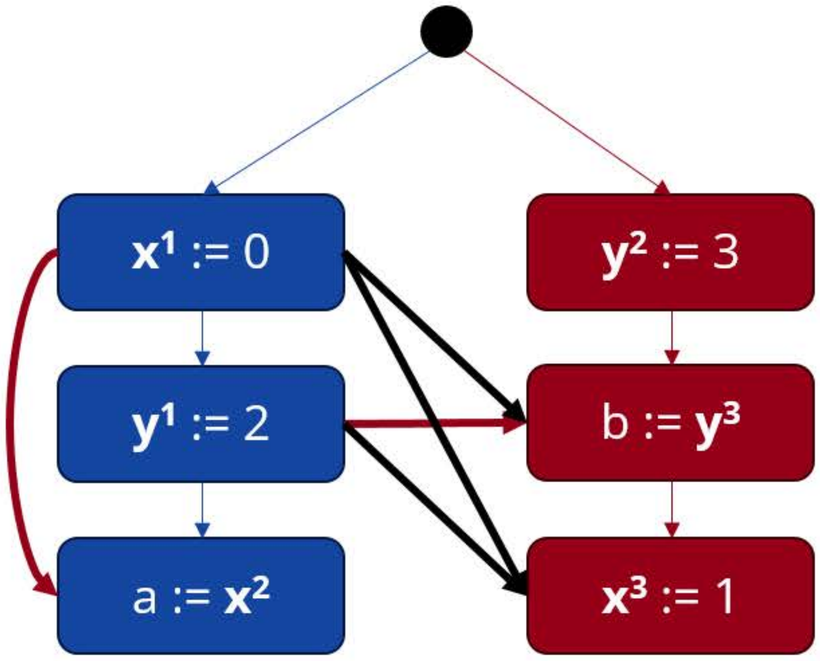
$x_1 \prec y_3$

Consistency-Preserving Propagation



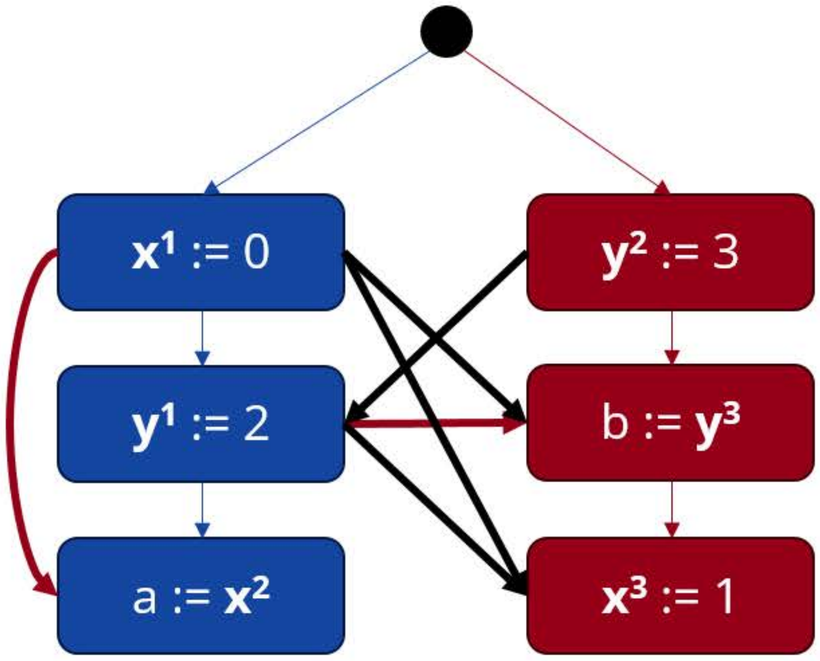
propagate($rf^x_{1,2}$)	$x_1 \prec x_2$ $x_1 \prec_{rf} x_2$
transitivity(x_1, x_2)	
derive(x_1, x_3, x_2)	
propagate($rf^y_{1,3}$)	$y_1 \prec y_3$ $y_1 \prec_{rf} y_3$
transitivity(y_1, y_3)	$y_1 \prec x_3$
transitivity(y_1, x_3)	$x_1 \prec x_3$
transitivity(x_1, x_3)	
	$x_1 \prec y_3$
transitivity(x_1, y_3)	

Consistency-Preserving Propagation



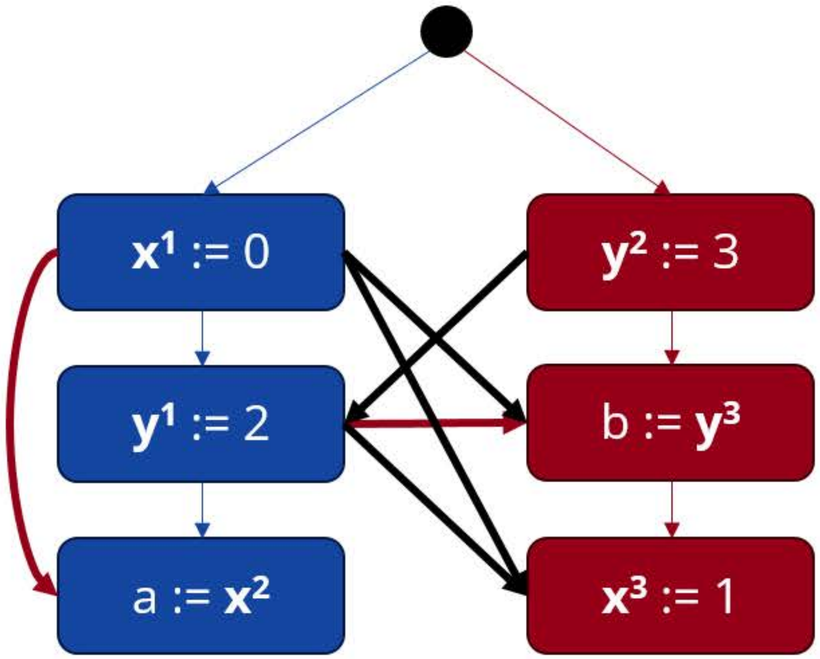
propagate($rf^x_{1,2}$)	$x_1 \prec x_2$ $x_1 \prec_{rf} x_2$
transitivity(x_1, x_2)	
derive(x_1, x_3, x_2)	
propagate($rf^y_{1,3}$)	$y_1 \prec y_3$ $y_1 \prec_{rf} y_3$
transitivity(y_1, y_3)	$y_1 \prec x_3$
transitivity(y_1, x_3)	$x_1 \prec x_3$
transitivity(x_1, x_3)	
	$x_1 \prec y_3$
transitivity(x_1, y_3)	
derive(y_1, y_2, y_3)	$y_2 \prec y_1$

Consistency-Preserving Propagation



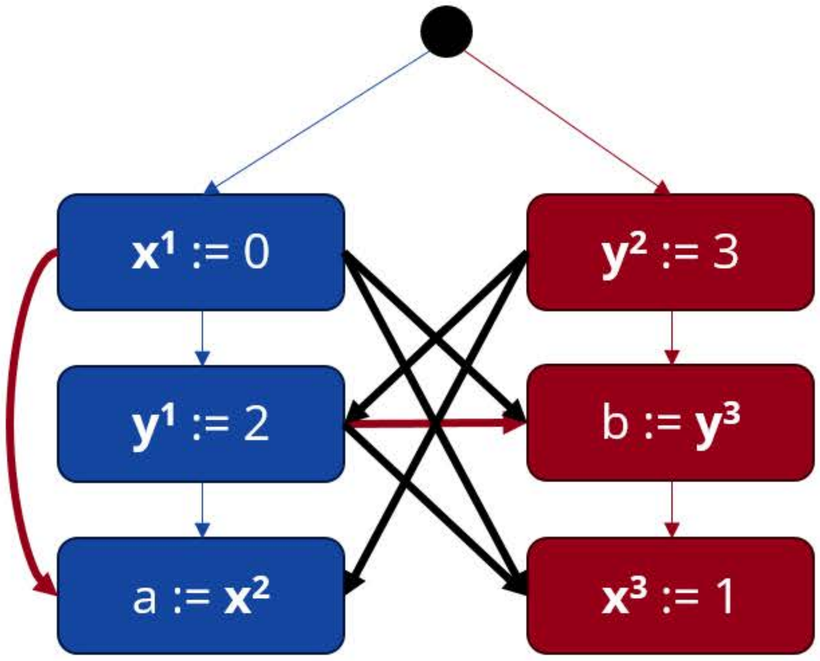
propagate($rf^x_{1,2}$)	$x_1 \prec x_2$ $x_1 \prec_{rf} x_2$
transitivity(x_1, x_2)	
derive(x_1, x_3, x_2)	
propagate($rf^y_{1,3}$)	$y_1 \prec y_3$ $y_1 \prec_{rf} y_3$
transitivity(y_1, y_3)	$y_1 \prec x_3$
transitivity(y_1, x_3)	$x_1 \prec x_3$
transitivity(x_1, x_3)	
	$x_1 \prec y_3$
transitivity(x_1, y_3)	
derive(y_1, y_2, y_3)	$y_2 \prec y_1$

Consistency-Preserving Propagation



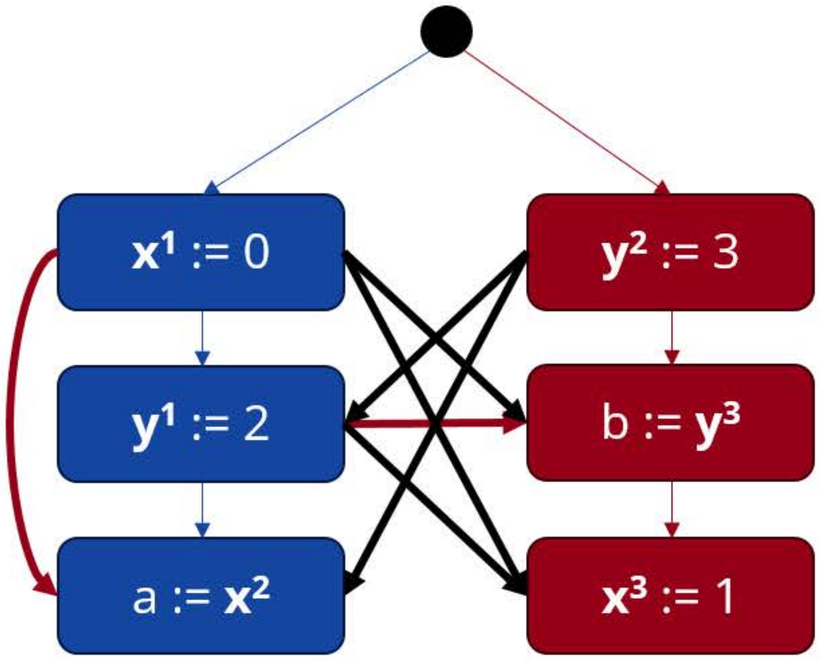
propagate($rf^x_{1,2}$)	$x_1 \prec x_2$	$x_1 \prec_{rf} x_2$
transitivity(x_1, x_2)		
derive(x_1, x_3, x_2)		
propagate($rf^y_{1,3}$)	$y_1 \prec y_3$	$y_1 \prec_{rf} y_3$
transitivity(y_1, y_3)	$y_1 \prec x_3$	
transitivity(y_1, x_3)	$x_1 \prec x_3$	
transitivity(x_1, x_3)		
	$x_1 \prec y_3$	
transitivity(x_1, y_3)		
derive(y_1, y_2, y_3)	$y_2 \prec y_1$	
transitivity(y_2, y_1)	$y_2 \prec x_2$	

Consistency-Preserving Propagation



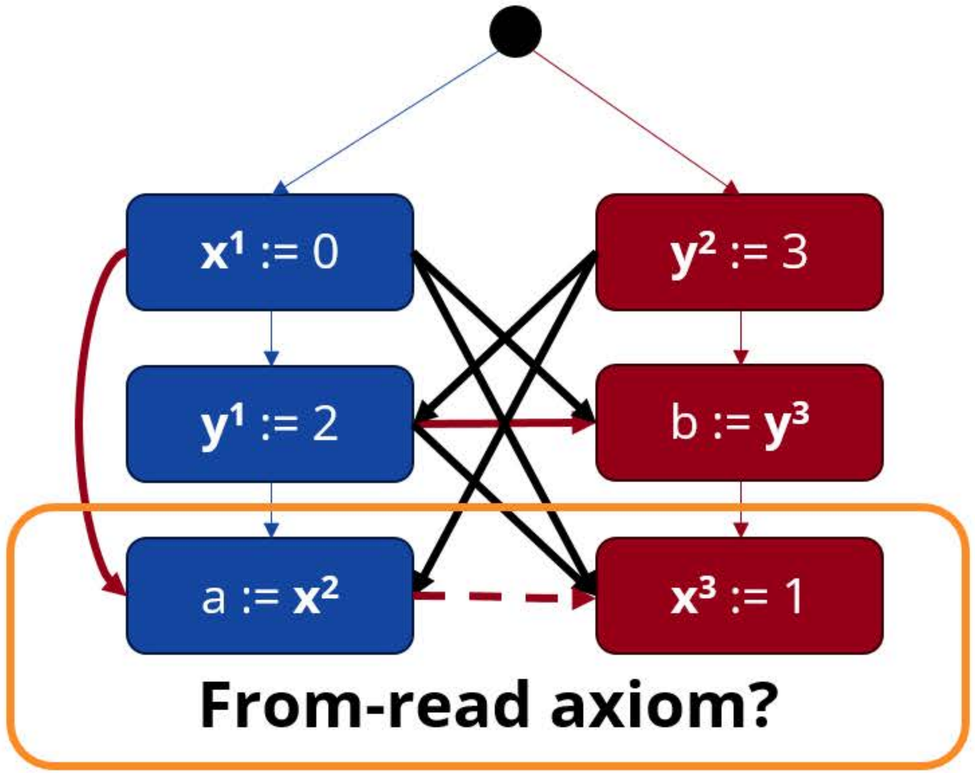
propagate($rf^x_{1,2}$)	$x_1 \prec x_2$	$x_1 \prec_{rf} x_2$
transitivity(x_1, x_2)		
derive(x_1, x_3, x_2)		
propagate($rf^y_{1,3}$)	$y_1 \prec y_3$	$y_1 \prec_{rf} y_3$
transitivity(y_1, y_3)	$y_1 \prec x_3$	
transitivity(y_1, x_3)	$x_1 \prec x_3$	
transitivity(x_1, x_3)		
	$x_1 \prec y_3$	
transitivity(x_1, y_3)		
derive(y_1, y_2, y_3)	$y_2 \prec y_1$	
transitivity(y_2, y_1)	$y_2 \prec x_2$	

Consistency-Preserving Propagation



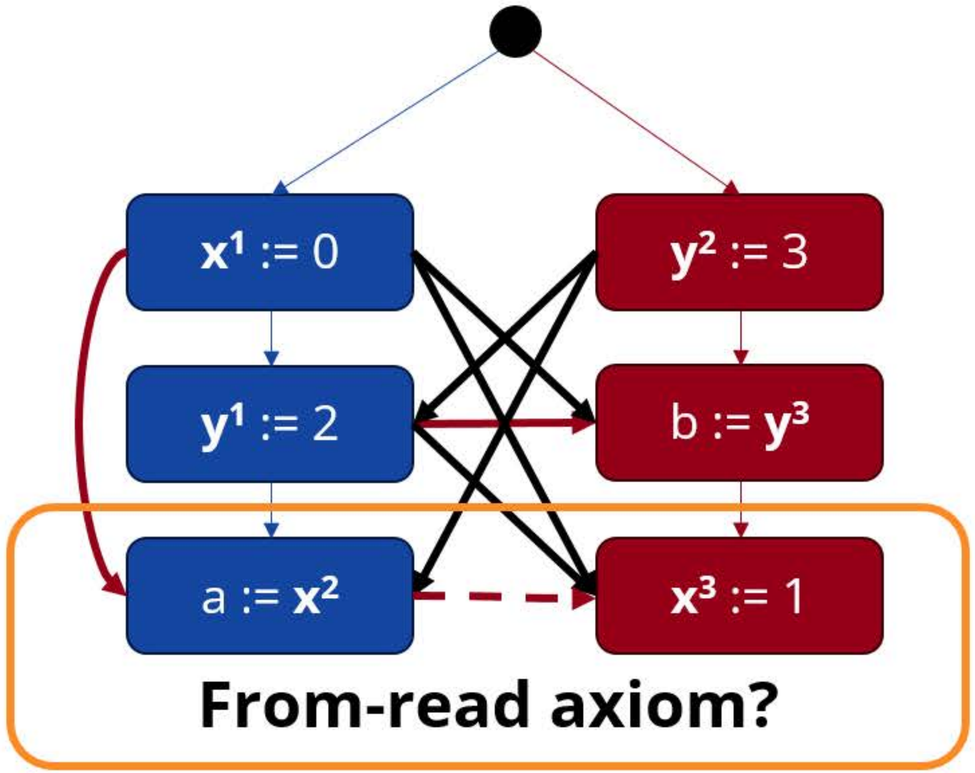
propagate($rf^x_{1,2}$)	$x_1 \prec x_2$	$x_1 \prec_{rf} x_2$
transitivity(x_1, x_2)		
derive(x_1, x_3, x_2)		
propagate($rf^y_{1,3}$)	$y_1 \prec y_3$	$y_1 \prec_{rf} y_3$
transitivity(y_1, y_3)	$y_1 \prec x_3$	
transitivity(y_1, x_3)	$x_1 \prec x_3$	
transitivity(x_1, x_3)		
	$x_1 \prec y_3$	
transitivity(x_1, y_3)		
derive(y_1, y_2, y_3)	$y_2 \prec y_1$	
transitivity(y_2, y_1)	$y_2 \prec x_2$	
transitivity(y_2, x_2)		

Consistency-Preserving Propagation



propagate($rf^x_{1,2}$)	$x_1 \prec x_2$ $\quad x_1 \prec_{rf} x_2$
transitivity(x_1, x_2)	
derive(x_1, x_3, x_2)	
propagate($rf^y_{1,3}$)	$y_1 \prec y_3$ $\quad y_1 \prec_{rf} y_3$
transitivity(y_1, y_3)	$y_1 \prec x_3$
transitivity(y_1, x_3)	$x_1 \prec x_3$
transitivity(x_1, x_3)	
	$x_1 \prec y_3$
transitivity(x_1, y_3)	
derive(y_1, y_2, y_3)	$y_2 \prec y_1$
transitivity(y_2, y_1)	$y_2 \prec x_2$
transitivity(y_2, x_2)	

Consistency-Preserving Propagation



propagate($rf^x_{1,2}$)	$x_1 \prec x_2$ $\quad x_1 \prec_{rf} x_2$
transitivity(x_1, x_2)	
derive(x_1, x_3, x_2)	
propagate($rf^y_{1,3}$)	$y_1 \prec y_3$ $\quad y_1 \prec_{rf} y_3$
transitivity(y_1, y_3)	$y_1 \prec x_3$
transitivity(y_1, x_3)	$x_1 \prec x_3$
transitivity(x_1, x_3)	
	$x_1 \prec y_3$
transitivity(x_1, y_3)	
derive(y_1, y_2, y_3)	$y_2 \prec y_1$
transitivity(y_2, y_1)	$y_2 \prec x_2$
transitivity(y_2, x_2)	

Stability is **not** preserved!

Propagation for Sequential Consistency

Propagation for Sequential Consistency

- SC: all memory instructions preserve source ordering

Propagation for Sequential Consistency

- SC: all memory instructions preserve source ordering
- Write-Read pairing (VSC-Read problem): **NP-complete**

Propagation for Sequential Consistency

- SC: all memory instructions preserve source ordering
- Write-Read pairing (VSC-Read problem): **NP-complete**
- The algorithm:
  - **Linear** in rf-assignments!

Algorithm 1 Original Theory Propagation Algorithm [5], with typos fixed

Input: l: Positive literal

```
proc PROPAGATE(l)
    if l is rf_ij then
        w_i, r_j := write and read of rf_ij
        ≺_rf ← ≺_rf ∪ {(w_i, r_j)}
        ≺ ← ≺ ∪ {(w_i, r_j)}
        TRANSITIVITY(w_i, r_j)
        foreach w_k s.t. grd_k do
            if (w_i, w_k, r_j) same-addr. then
                DERIVE(w_i, w_k, r_j)
            end if
        end foreach
    else if l is grd_k then
        w_k := write of grd_k
        foreach (w_i, r_i) ∈ ≺_rf do
            if (w_i, w_k, r_j) same-addr. then
                DERIVE(w_i, w_k, r_j)
            end if
        end foreach
    end if
end proc
```

Input: w_i, w_k, r_j: $w_i \prec_{rf} r_j, grd_k$

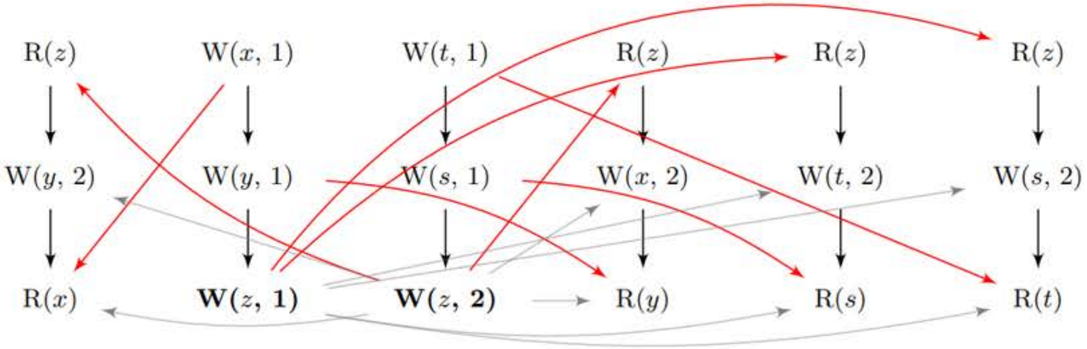
```
proc DERIVE(w_i, w_k, r_j)
    if w_k ≺ r_j then
        ≺ ← ≺ ∪ {(w_k, w_i)}
        TRANSITIVITY(w_k, w_i)
    else if w_i ≺ w_k then
        ≺ ← ≺ ∪ {(r_j, w_k)}
        TRANSITIVITY(r_j, w_k)
    end if
end proc
```

Input: e_1, e_2: $e_1 \prec e_2$

```
proc TRANSITIVITY(e_1, e_2)
    foreach (e_2, e_3) ∈ ≺ do
        if (e_1, e_3) ∉ ≺ then
            ≺ ← ≺ ∪ {(e_1, e_3)}
            TRANSITIVITY(e_1, e_3)
        end if
    end foreach
    foreach (e_0, e_1) ∈ ≺ do
        if (e_0, e_2) ∉ ≺ then
            ≺ ← ≺ ∪ {(e_0, e_2)}
            TRANSITIVITY(e_0, e_2)
        end if
    end foreach
end proc
```

Propagation for Sequential Consistency

- SC: all memory instructions preserve source ordering
- Write-Read pairing (VSC-Read problem): **NP-complete**
- The algorithm:
  - **Linear** in rf-assignments!



Algorithm 1 Original Theory Propagation Algorithm [5], with typos fixed

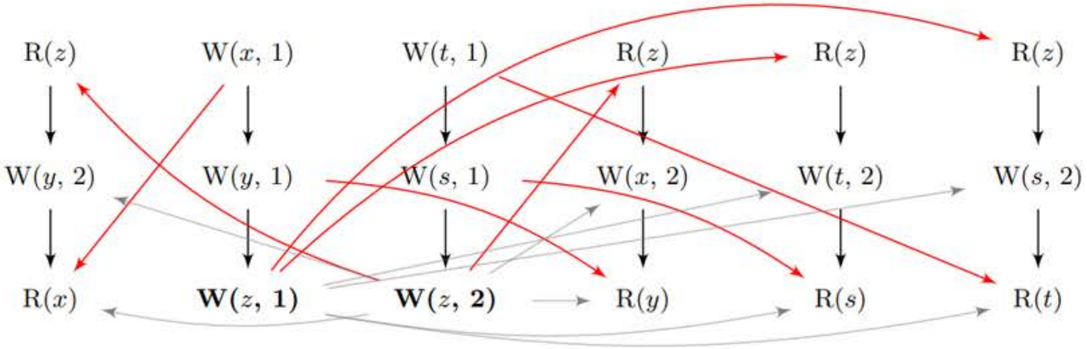
Input: l: Positive literal
proc PROPAGATE(l)
  if l is rf_{ij} **then**
    w_i, r_j := write and read of rf_{ij}
    $\prec_{rf} \leftarrow \prec_{rf} \cup \{(w_i, r_j)\}$
    $\prec \leftarrow \prec \cup \{(w_i, r_j)\}$
    TRANSITIVITY(w_i, r_j)
    foreach w_k *s.t.* grd_k **do**
      if (w_i, w_k, r_j) same-addr. **then**
        DERIVE(w_i, w_k, r_j)
      end if
    end foreach
  else if l is grd_k **then**
    w_k := write of grd_k
    foreach $(w_i, r_i) \in \prec_{rf}$ **do**
      if (w_i, w_k, r_j) same-addr. **then**
        DERIVE(w_i, w_k, r_j)
      end if
    end foreach
  end if
end proc

Input: w_i, w_k, r_j: $w_i \prec_{rf} r_j, grd_k$
proc DERIVE(w_i, w_k, r_j)
  if $w_k \prec r_j$ **then**
    $\prec \leftarrow \prec \cup \{(w_k, w_i)\}$
    TRANSITIVITY(w_k, w_i)
  else if $w_i \prec w_k$ **then**
    $\prec \leftarrow \prec \cup \{(r_j, w_k)\}$
    TRANSITIVITY(r_j, w_k)
  end if
end proc

Input: e_1, e_2: $e_1 \prec e_2$
proc TRANSITIVITY(e_1, e_2)
  foreach $(e_2, e_3) \in \prec$ **do**
    if $(e_1, e_3) \notin \prec$ **then**
      $\prec \leftarrow \prec \cup \{(e_1, e_3)\}$
      TRANSITIVITY(e_1, e_3)
    end if
  end foreach
  foreach $(e_0, e_1) \in \prec$ **do**
    if $(e_0, e_2) \notin \prec$ **then**
      $\prec \leftarrow \prec \cup \{(e_0, e_2)\}$
      TRANSITIVITY(e_0, e_2)
    end if
  end foreach
end proc

Propagation for Sequential Consistency

- SC: all memory instructions preserve source ordering
- Write-Read pairing (VSC-Read problem): **NP-complete**
- The algorithm:
  - **Linear** in rf-assignments!



Weak Sequential Consistency

Algorithm 1 Original Theory Propagation Algorithm [5], with typos fixed

Input: l: Positive literal
proc PROPAGATE(l)
  if l is rf_{ij} **then**
    w_i, r_j := write and read of rf_{ij}
    $\prec_{rf} \leftarrow \prec_{rf} \cup \{(w_i, r_j)\}$
    $\prec \leftarrow \prec \cup \{(w_i, r_j)\}$
    TRANSITIVITY(w_i, r_j)
    foreach w_k *s.t.* grd_k **do**
      if (w_i, w_k, r_j) same-addr. **then**
        DERIVE(w_i, w_k, r_j)
      end if
    end foreach
  else if l is grd_k **then**
    w_k := write of grd_k
    foreach $(w_i, r_i) \in \prec_{rf}$ **do**
      if (w_i, w_k, r_j) same-addr. **then**
        DERIVE(w_i, w_k, r_j)
      end if
    end foreach
  end if
end proc

Input: w_i, w_k, r_j: $w_i \prec_{rf} r_j, grd_k$
proc DERIVE(w_i, w_k, r_j)
  if $w_k \prec r_j$ **then**
    $\prec \leftarrow \prec \cup \{(w_k, w_i)\}$
    TRANSITIVITY(w_k, w_i)
  else if $w_i \prec w_k$ **then**
    $\prec \leftarrow \prec \cup \{(r_j, w_k)\}$
    TRANSITIVITY(r_j, w_k)
  end if
end proc

Input: e_1, e_2: $e_1 \prec e_2$
proc TRANSITIVITY(e_1, e_2)
  foreach $(e_2, e_3) \in \prec$ **do**
    if $(e_1, e_3) \notin \prec$ **then**
      $\prec \leftarrow \prec \cup \{(e_1, e_3)\}$
      TRANSITIVITY(e_1, e_3)
    end if
  end foreach
  foreach $(e_0, e_1) \in \prec$ **do**
    if $(e_0, e_2) \notin \prec$ **then**
      $\prec \leftarrow \prec \cup \{(e_0, e_2)\}$
      TRANSITIVITY(e_0, e_2)
    end if
  end foreach
end proc

Summary of problems

Summary of problems

- Stability claim: due to **propagation algorithm**

Summary of problems

- Stability claim: due to **propagation algorithm**
  - Causes happens-before orders to be **missed**

Summary of problems

- Stability claim: due to **propagation algorithm**
  - Causes happens-before orders to be **missed**
  - May accept results with cyclic happens-before order

Summary of problems

- Stability claim: due to **propagation algorithm**
  - Causes happens-before orders to be **missed**
  - May accept results with cyclic happens-before order
  - May lead to **false positive results**

Summary of problems

- Stability claim: due to **propagation algorithm**
  - Causes happens-before orders to be **missed**
  - May accept results with cyclic happens-before order
  - May lead to **false positive results**

- SC instead of WSC: due to **axioms**

Summary of problems

- Stability claim: due to **propagation algorithm**
  - Causes happens-before orders to be **missed**
  - May accept results with cyclic happens-before order
  - May lead to **false positive results**

- SC instead of WSC: due to **axioms**
  - Causes happens-before orders to be **unanalyzed**

Summary of problems

- Stability claim: due to **propagation algorithm**
  - Causes happens-before orders to be **missed**
  - May accept results with cyclic happens-before order
  - May lead to **false positive results**

- SC instead of WSC: due to **axioms**
  - Causes happens-before orders to be **unanalyzed**
  - Possible that no order exists

Summary of problems

- Stability claim: due to **propagation algorithm**
  - Causes happens-before orders to be **missed**
  - May accept results with cyclic happens-before order
  - May lead to **false positive results**

- SC instead of WSC: due to **axioms**
  - Causes happens-before orders to be **unanalyzed**
  - Possible that no order exists
  - May lead to **false positive results**

Summary of problems

- Stability claim: due to **propagation algorithm**
  - Causes happens-before orders to be **missed**
  - May accept results with cyclic happens-before order
  - May lead to **false positive results**

- SC instead of WSC: due to **axioms**
  - Causes happens-before orders to be **unanalyzed**
  - Possible that no order exists
  - May lead to **false positive results**
  - **Will not fix –** we want to fix the **algorithm**, not the **axioms**

Summary of problems

- Stability claim: due to **propagation algorithm**
  - Causes happens-before orders to be **missed**
  - May accept results with cyclic happens-before order
  - May lead to **false positive results**

- SC instead of WSC: due to **axioms**
  - Causes happens-before orders to be **unanalyzed**
  - Possible that no order exists
  - May lead to **false positive results**
  - **Will not fix –** we want to fix the **algorithm**, not the **axioms**
    - Easily fixable to allow SC: try to order all writes post-propagation

Fixing Stability

Retrospectively

Repeat PROPAGATE until no new changes occur

Proof of Stability

Prospectively

Find issue and fix it

Proof of Stability

Experimental Evaluation

Experimental Evaluation

RQ1: Performance differences between original, retrospective, and prospective algorithms in isolation.

Experimental Evaluation

RQ1: Performance differences between original, retrospective, and prospective algorithms in isolation.

RQ2: How Deagle avoids false results despite flaws

Experimental Evaluation

RQ1: Performance differences between original, retrospective, and prospective algorithms in isolation.

RQ2: How Deagle avoids false results despite flaws

RQ3: Transfer of revised algorithms' performance to Theta.

Experimental Evaluation

RQ1: Performance differences between original, retrospective, and prospective algorithms in isolation.

RQ2: How Deagle avoids false results despite flaws

RQ3: Transfer of revised algorithms' performance to Theta.

RQ4: Performance of retrospective vs. prospective algorithms in Theta

Isolated implementation

Isolated implementation

- SV-BENCHMARKS, CONCURRENCYSAFETY, unrolled twice

Isolated implementation

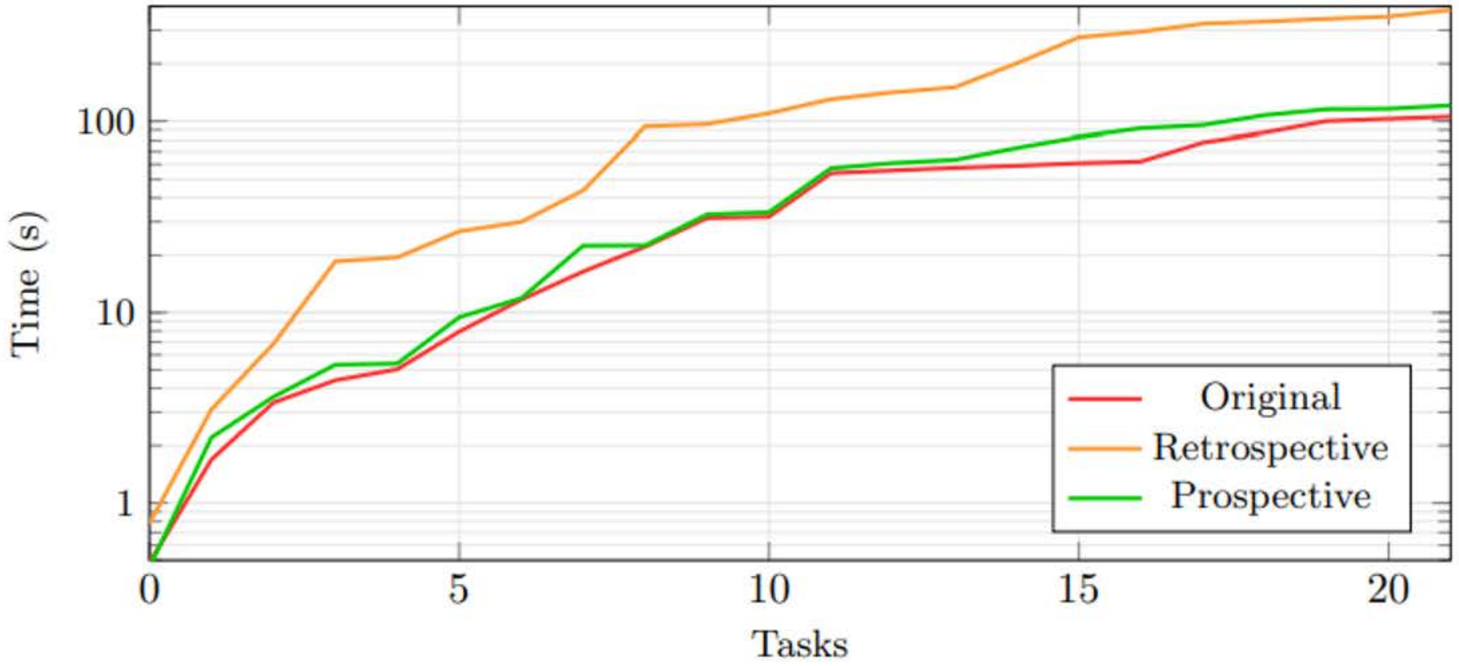
- SV-BENCHMARKS, CONCURRENCYSAFETY, unrolled twice
- Original, retrospective, prospective configurations

Isolated implementation

- SV-BENCHMARKS, CONCURRENCYSAFETY, unrolled twice
- Original, retrospective, prospective configurations
- **22 tasks**: mismatch in happens-before after propagation

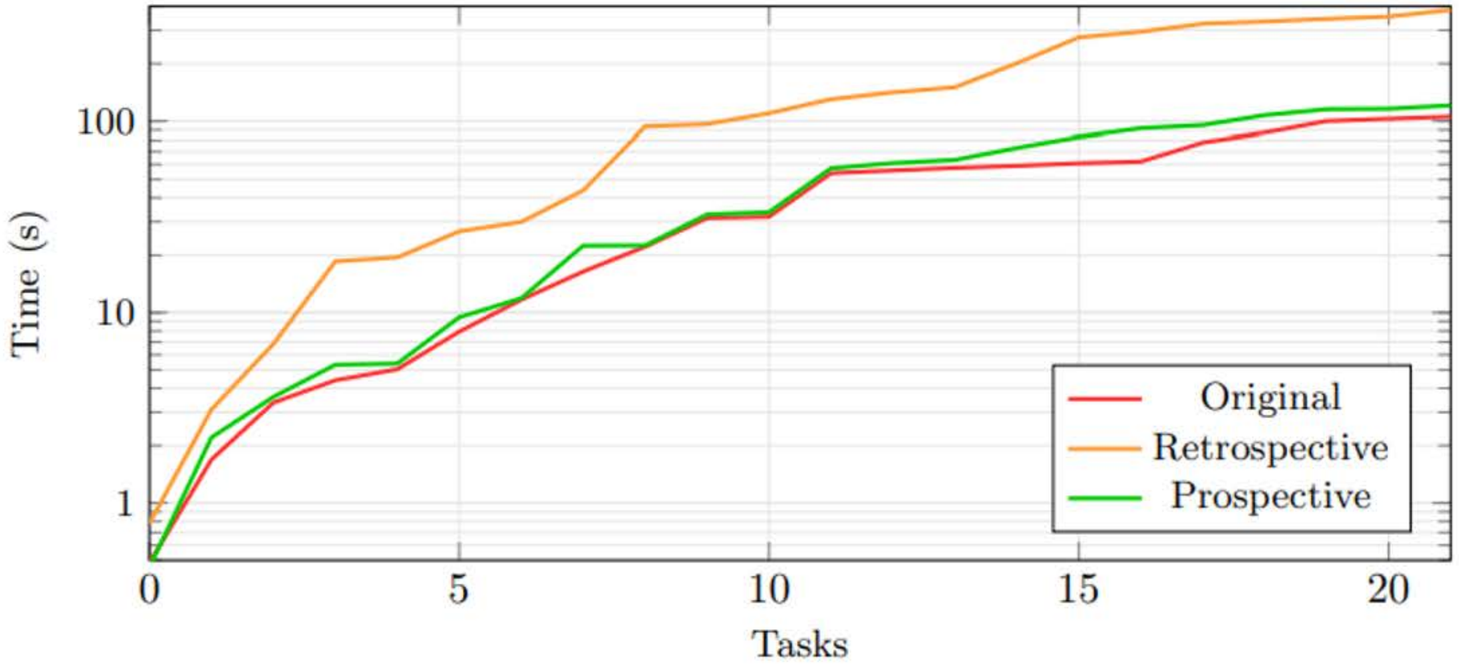
Isolated implementation

- SV-BENCHMARKS, CONCURRENCYSAFETY, unrolled twice
- Original, retrospective, prospective configurations
- **22 tasks**: mismatch in happens-before after propagation



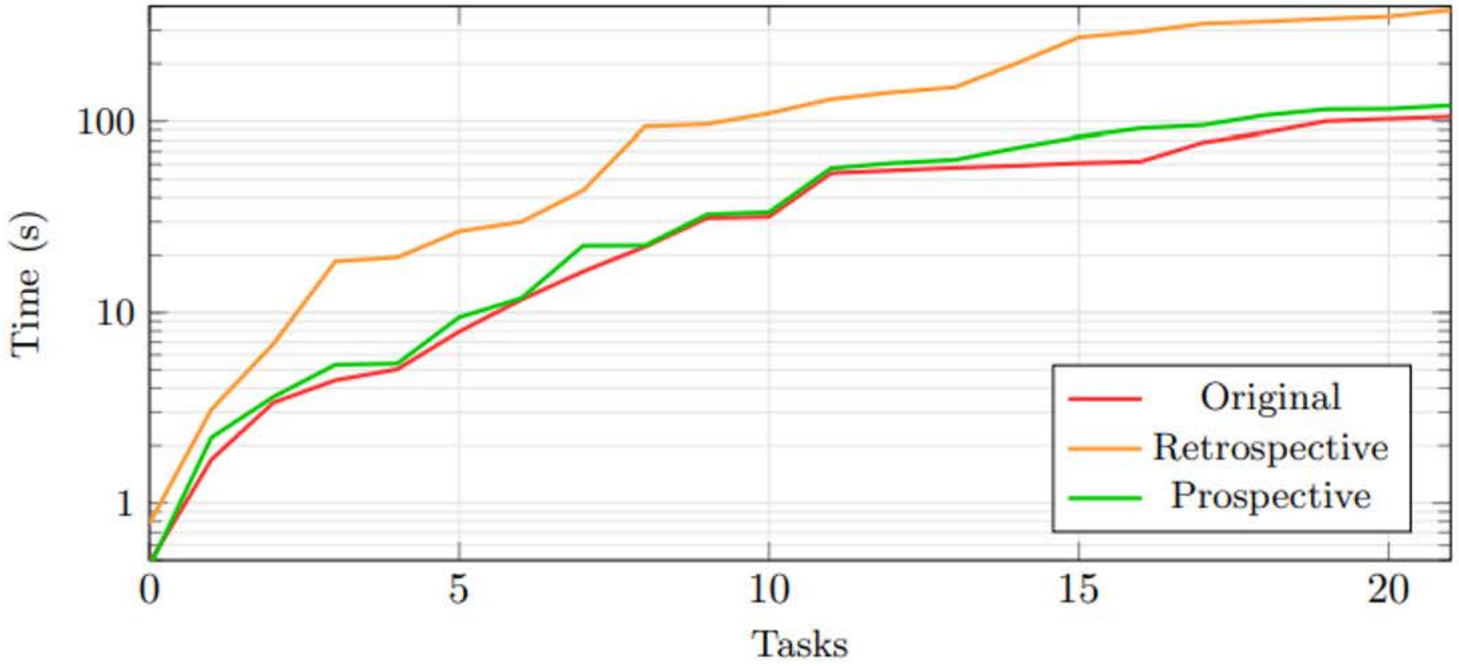
Isolated implementation

- SV-BENCHMARKS, CONCURRENCYSAFETY, unrolled twice
- Original, retrospective, prospective configurations
- **22 tasks**: mismatch in happens-before after propagation
  - Original is **fastest**



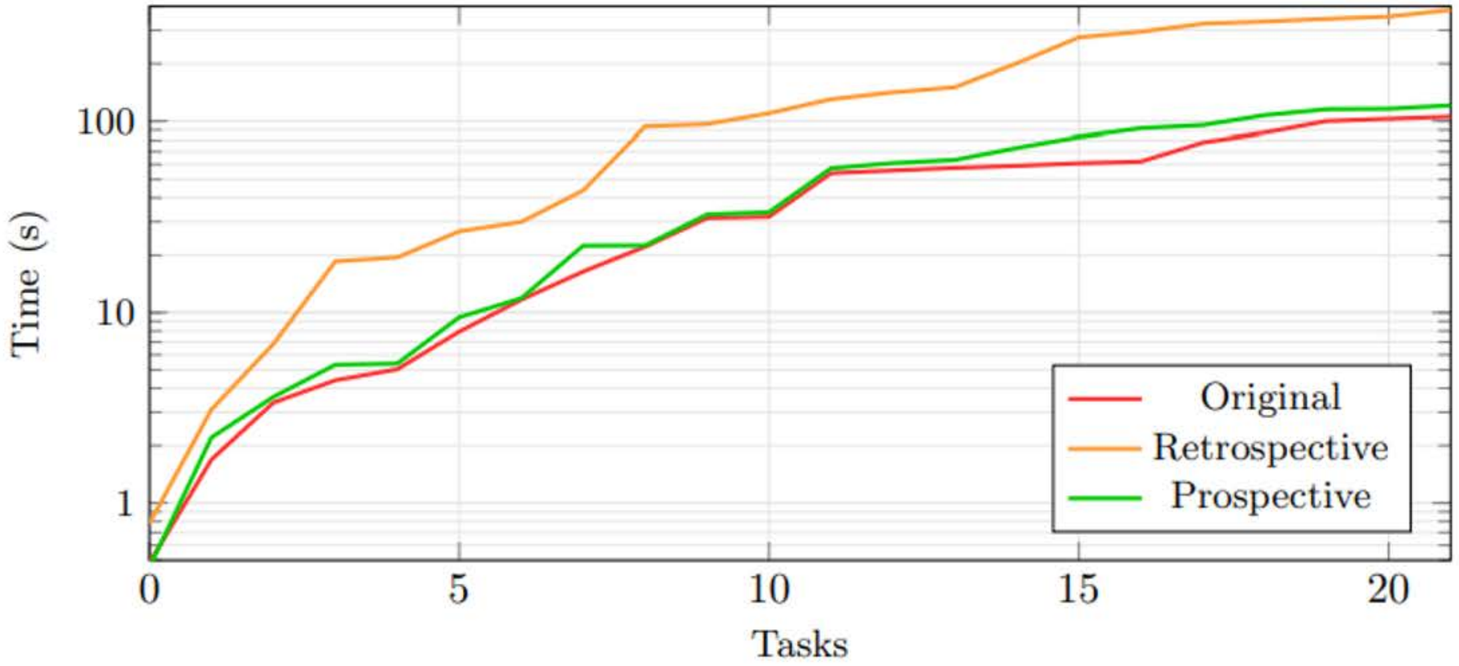
Isolated implementation

- SV-BENCHMARKS, CONCURRENCYSAFETY, unrolled twice
- Original, retrospective, prospective configurations
- **22 tasks**: mismatch in happens-before after propagation
  - Original is **fastest**
  - Prospective is close



Isolated implementation

- SV-BENCHMARKS, CONCURRENCYSAFETY, unrolled twice
- Original, retrospective, prospective configurations
- **22 tasks**: mismatch in happens-before after propagation
  - Original is **fastest**
  - Prospective is close
  - Retrospective is **much slower**



Deagle implementation

Deagle implementation

- Implementation not aligned with publication

Deagle implementation

- Implementation not aligned with publication
  - ONE_MORE_TIME flag: if something isn't ordered, re-do (~**retrospective**)

Deagle implementation

- Implementation not aligned with publication
  - ONE_MORE_TIME flag: if something isn't ordered, re-do (~**retrospective**)
  - **patched** to ignore

Deagle implementation

- Implementation not aligned with publication
  - ONE_MORE_TIME flag: if something isn't ordered, re-do (~**retrospective**)
  - **patched** to ignore

	binary release (retrospective)	source code (retrospective)	patched (original)
correct	**618**	**628**	**236**
true	320	332	165
false	298	296	71
incorrect	**0**	**14**	**20**
true	0	10	4
false	0	4	16

Deagle implementation

- Implementation not aligned with publication
  - ONE_MORE_TIME flag: if something isn't ordered, re-do (~**retrospective**)
  - **patched** to ignore

	binary release (retrospective)	source code (retrospective)	patched (original)
correct	**618**	**628**	**236**
true	320	332	165
false	298	296	71
incorrect	**0**	**14**	**20**
true	0	10	4
false	0	4	16

- Source version not aligned with SV-COMP'24 version

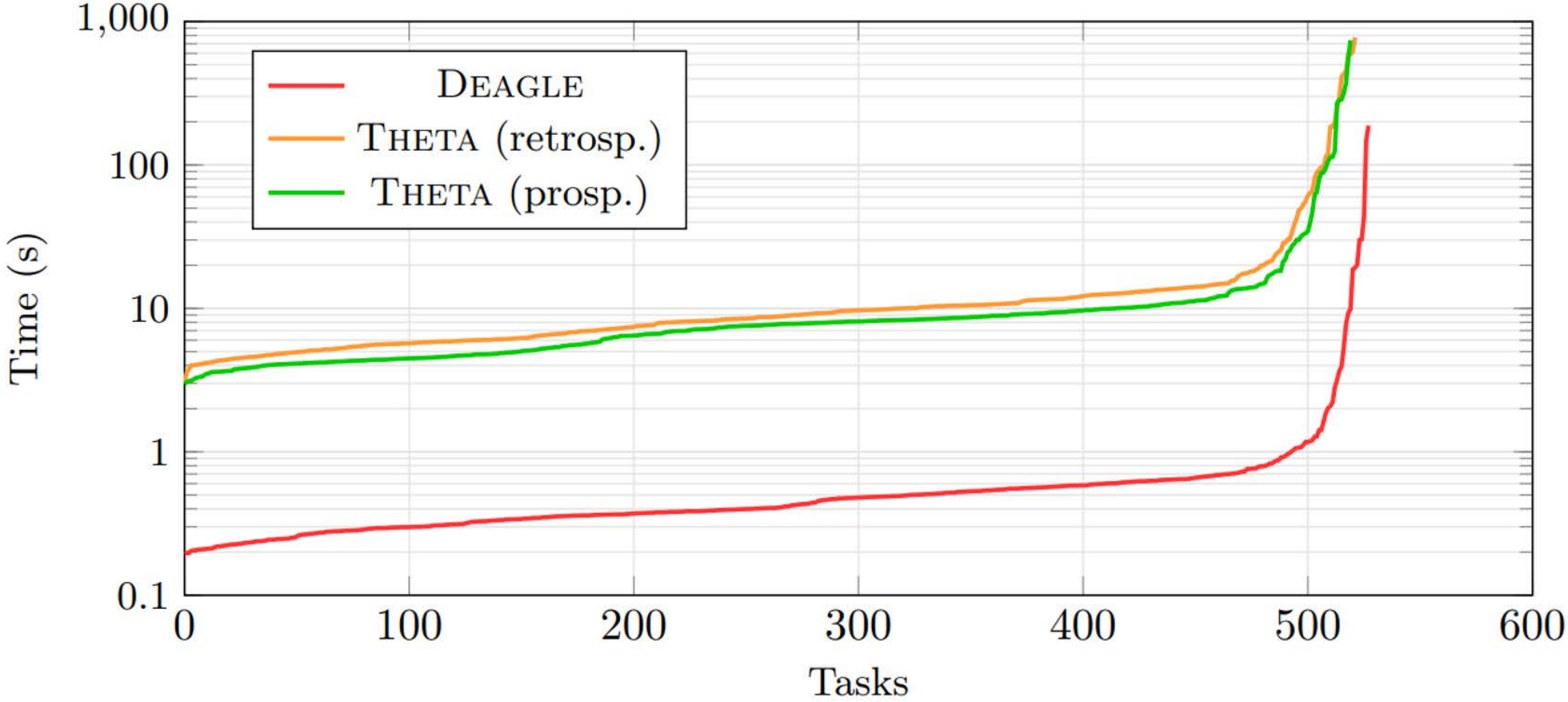
Deagle implementation

- Implementation not aligned with publication
  - ONE_MORE_TIME flag: if something isn't ordered, re-do (~**retrospective**)
  - **patched** to ignore

	binary release (retrospective)	source code (retrospective)	patched (original)
correct	**618**	**628**	**236**
true	320	332	165
false	298	296	71
incorrect	**0**	**14**	**20**
true	0	10	4
false	0	4	16

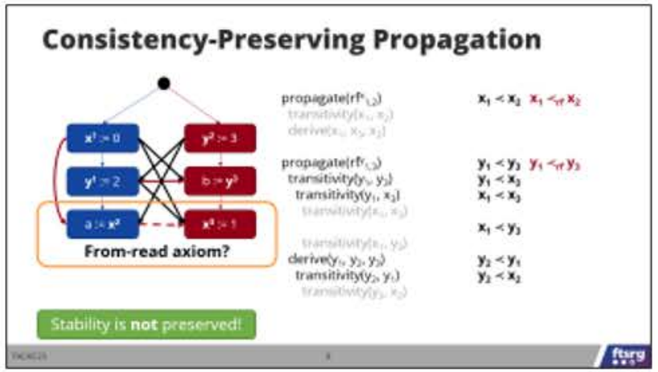
- Source version not aligned with SV-COMP'24 version
  - Mostly due to unrolling limits, not pushed to repo

Implementation in Theta



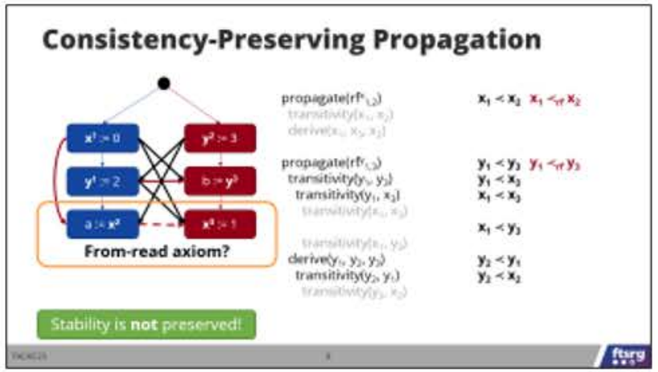
Summary of Contributions

Summary of Contributions

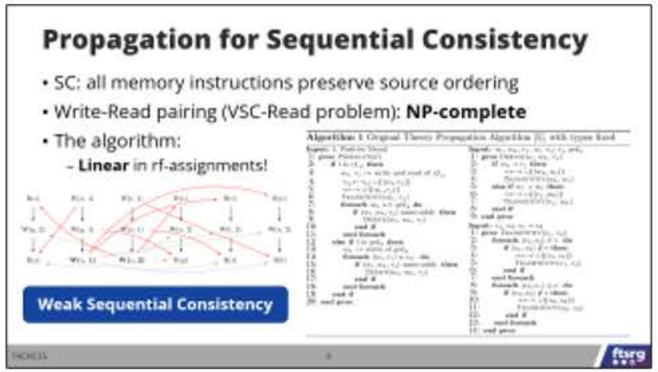


Shown CEx. to stability claim

Summary of Contributions

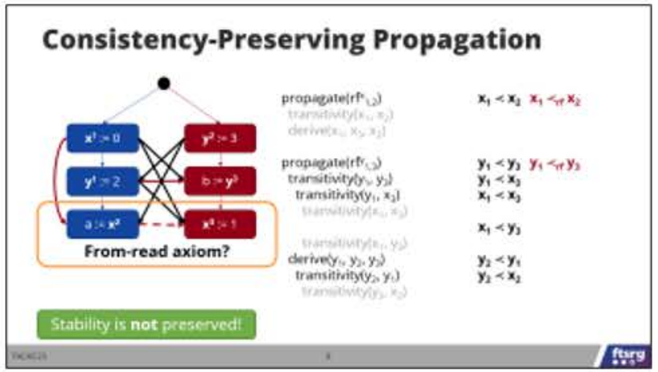


Shown CEx. to stability claim

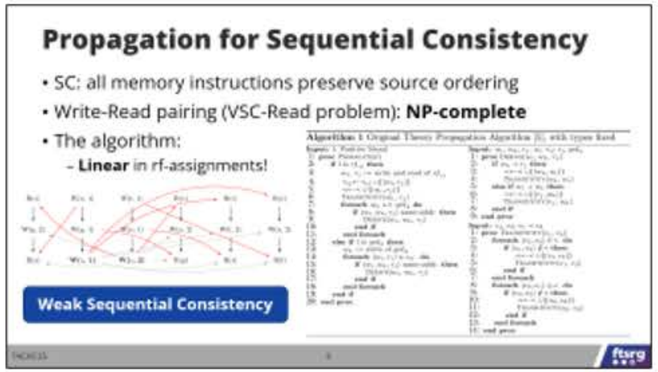


Shown inapplicability to SC

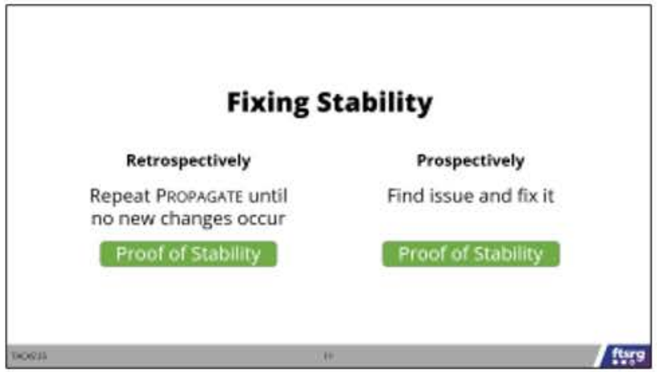
Summary of Contributions



Shown CEx. to stability claim

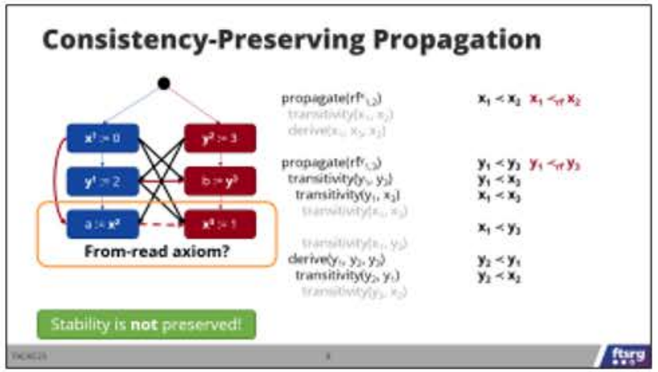


Shown inapplicability to SC

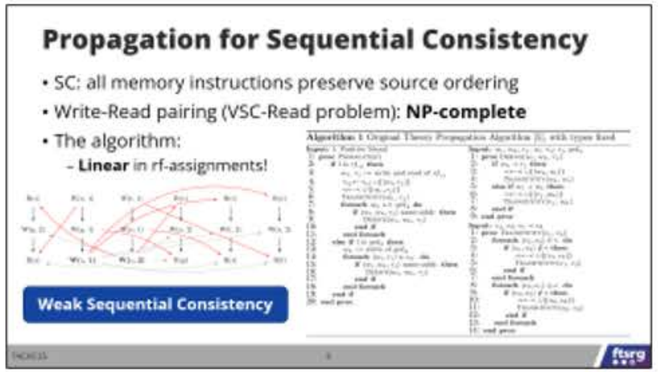


Fixed algorithm stability

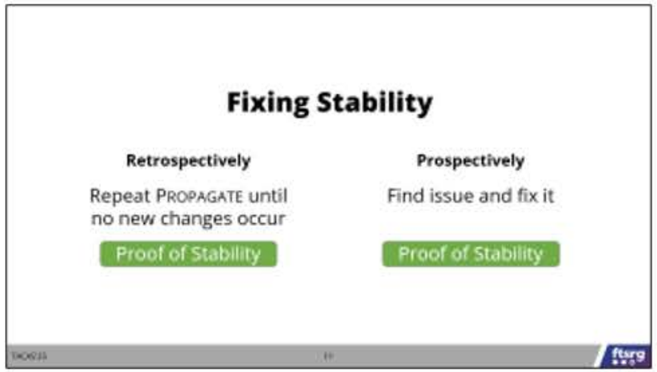
Summary of Contributions



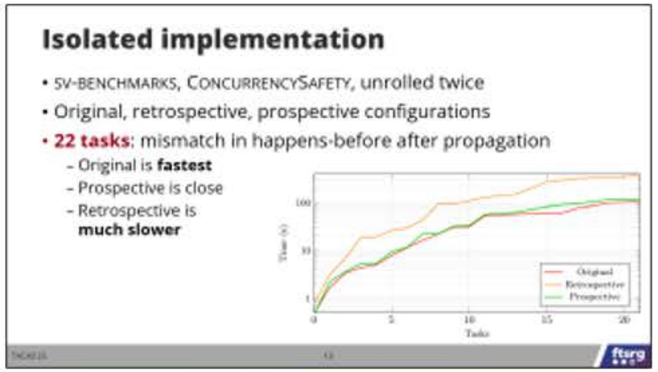
Shown CEx. to stability claim



Shown inapplicability to SC

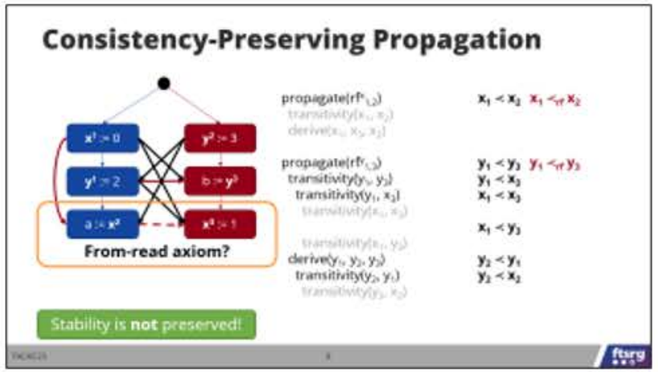


Fixed algorithm stability

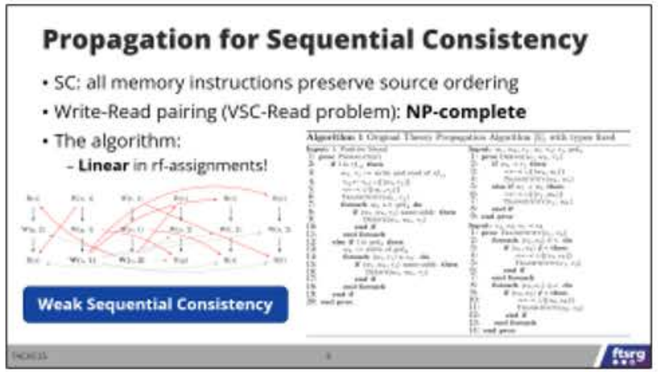


Evaluated in isolation

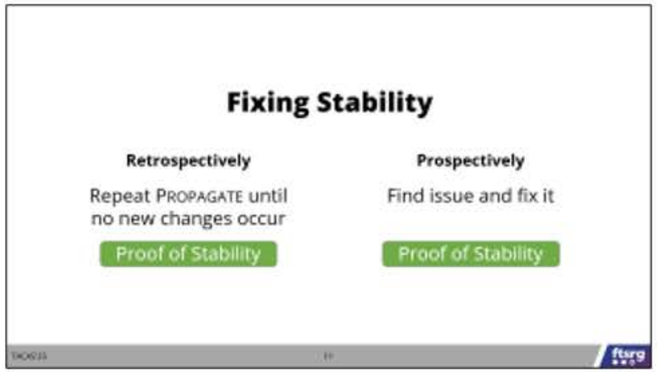
Summary of Contributions



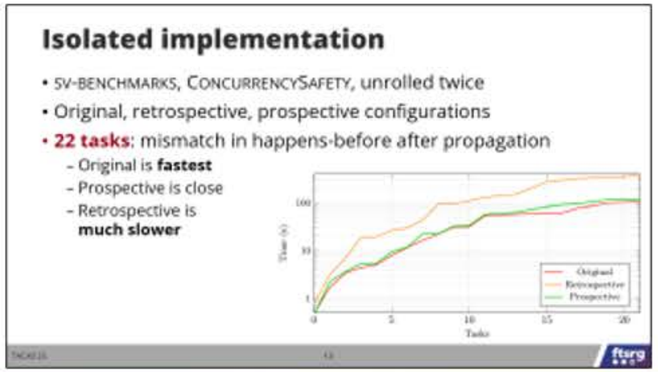
Shown CEx. to stability claim



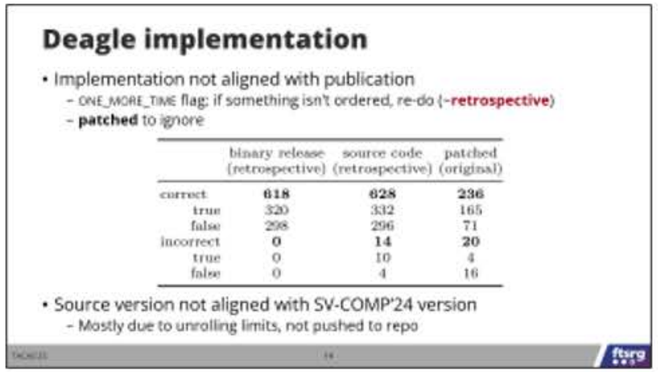
Shown inapplicability to SC



Fixed algorithm stability



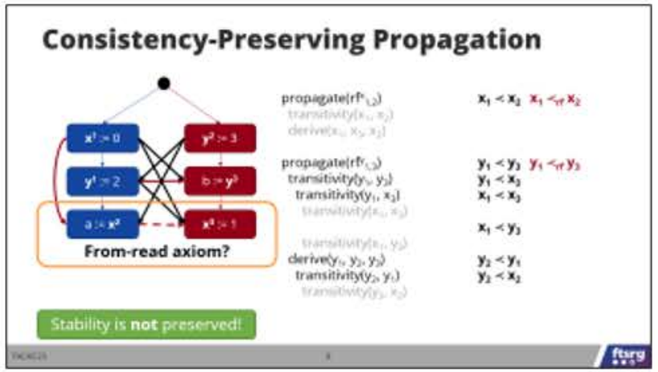
Evaluated in isolation



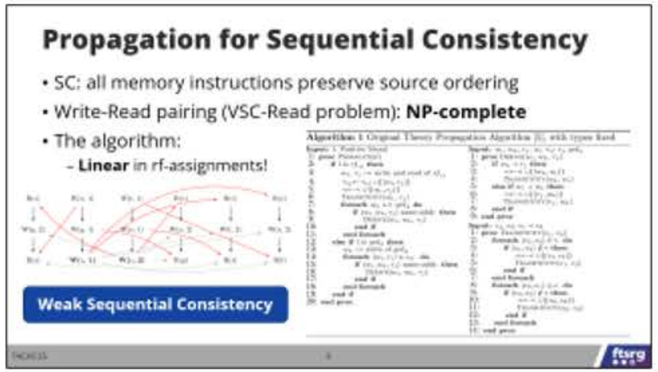
	binary release (retrospective)	source code (retrospective)	patched (original)
correct	**618**	**628**	**236**
true	320	332	165
false	298	296	71
incorrect	**0**	**14**	**20**
true	0	10	4
false	0	4	16

Analyzed Deagle impl.

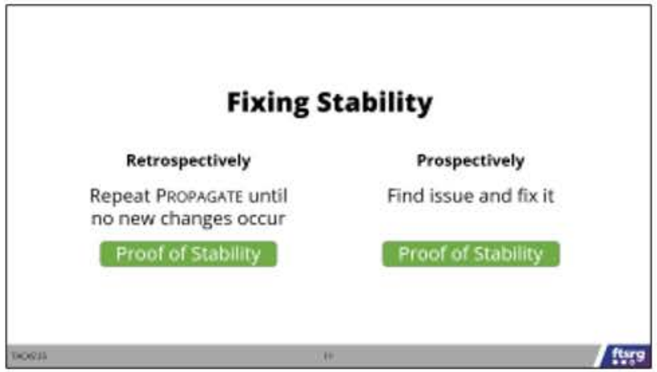
Summary of Contributions



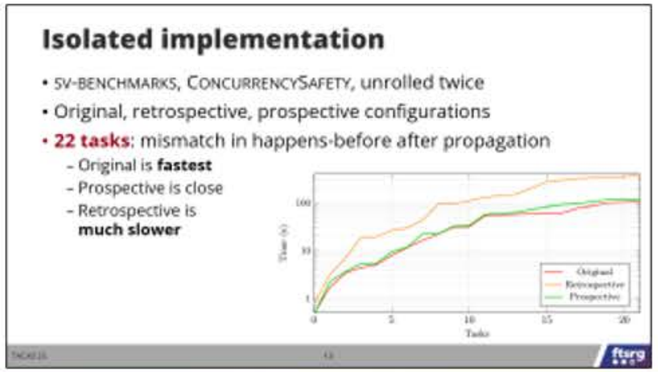
Shown CEx. to stability claim



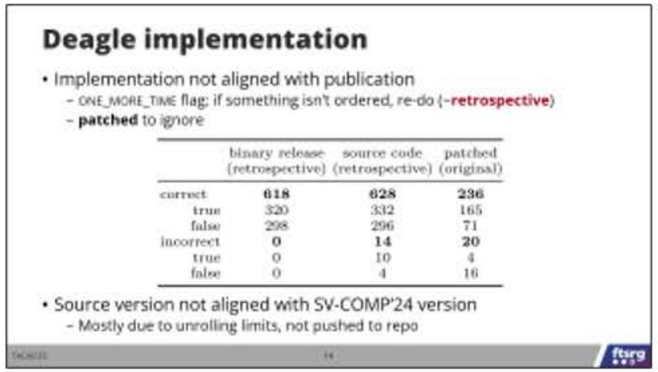
Shown inapplicability to SC



Fixed algorithm stability

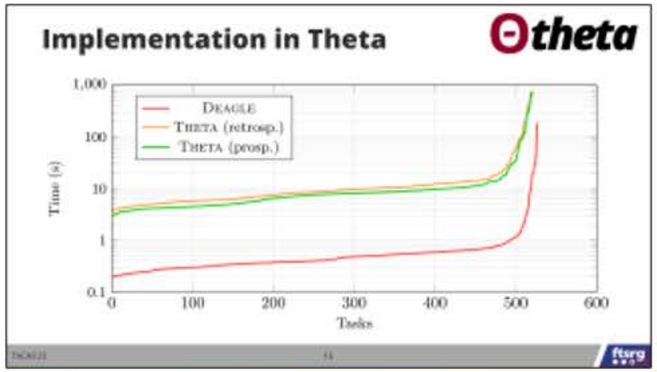


Evaluated in isolation



	binary release (retrospective)	source code (retrospective)	patched (original)
correct	**618**	**628**	**236**
true	320	332	165
false	298	296	71
incorrect	**0**	**14**	**20**
true	0	10	4
false	0	4	16

Analyzed Deagle impl.



Implemented in Theta