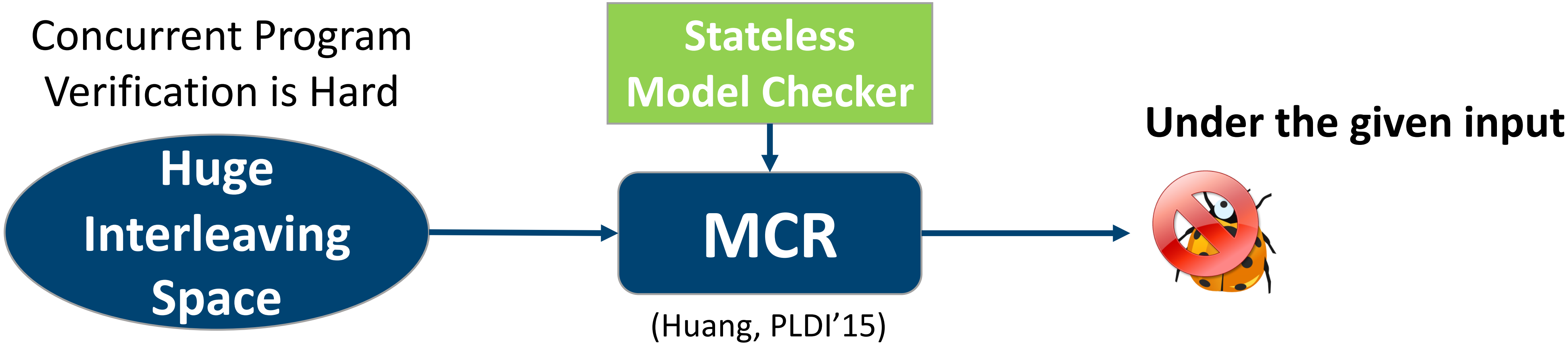
Speeding Up Maximal Causality Reduction with Static Analysis

Shiyou Huang   Jeff Huang
huangsy@tamu.edu
Parasol Lab, Texas A&M University

Parasol ASER
Automated Software Engineering Research
TEXAS A&M UNIVERSITY

Maximal Causality Reduction (MCR)

Concurrent Program Verification is Hard

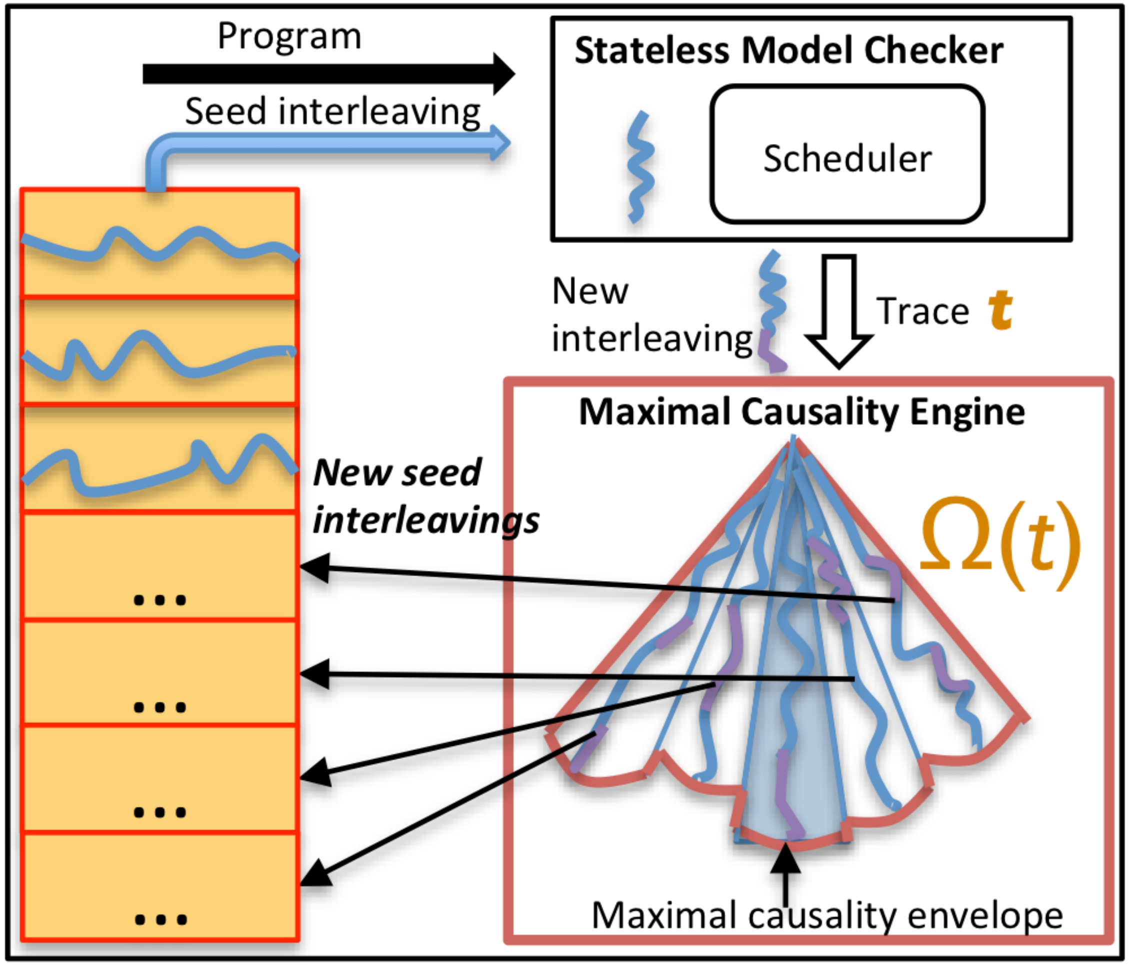


- + No redundancy
- + Sound and Complete
- + More efficient than DPOR[1] and ICB[2]

- Purely Dynamic, #constraints cubic in trace size
- Without considering input non-determinism

1. DPOR: Flanagan and Godefroid, PLDI'05
2. ICB: Musuvathi , OSDI'08

Maximal Causality Reduction (MCR)



(Huang, PLDI'15)

- Trace: A sequence of events executed by the program
- Constraints: An order variable (O) for each event in the trace
  E.g., if e1 happens before e2, $O_{e1} < O_{e2}$
- Interleaving: A sequence of thread schedule

Constraints Model -- $\boldsymbol{\Omega}(\mathbf{t})$

$$\Omega(t) = \phi_{mhb} \wedge \phi_{lock} \wedge \phi_{validity} \wedge \phi_{state}$$

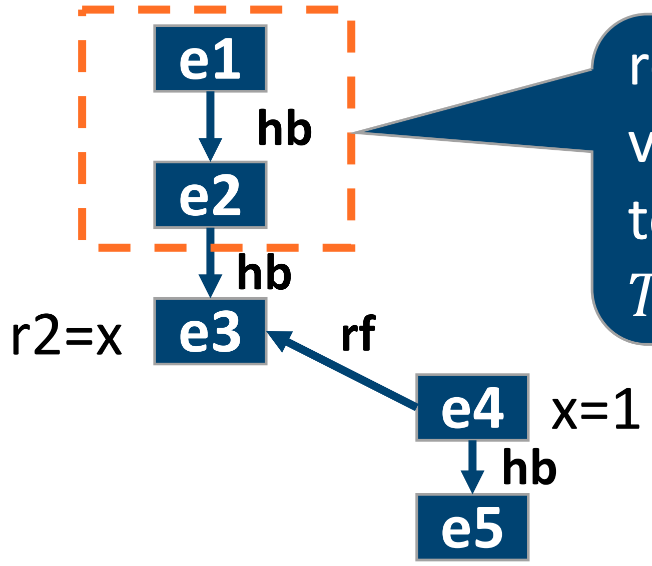
- **must-happen-before($\emptyset_{mhb}$)**
  E.g., $O1 < O2$ if e1 and e2 are by the same thread, and e1 occurs before e2
- **lock-mutual-exclusion($\emptyset_{lock}$)**
  E.g., for a lock pair, $(l1, u1)$ and $(l2, u2)$, $O_{u1} < O_{l2} \vee O_{u2} < O_{l1}$
- **validity($\emptyset_{validity}$)**
  an event is feasible if every read that must-happen-before it returns the same value
- **new state($\emptyset_{state}$)**
  At least one read in $\boldsymbol{t}$ returns a different value

An Example

$\mathbf{S0: 1\text{-}2\text{-}3\text{-}4\text{-}5},\ \mathrm{r1} = \mathrm{r2} = 0, True \equiv x == 0$

Init: x=y=0

T1	T2
1: r1=y	4: x=1
2: if (x==0)	5: y=1
3: r2=x	



return the same value as that in S0 to enforce $True \equiv x == 0$

Possible schedules:

1. 1-2-3-4-5
2. 1-2-4-3-5
3. 1-4-5-2
4. …

Constraints:

$HB: \quad e1 \prec e2 \prec e3,\ e4 \prec e5$

$State: e3 \prec e4$

$Validity: e1 \prec e5, e2 \prec e4$

4-1-2-3-5 ✗

$False \equiv x == 0$

1-2-4-3-5 ✔

Validity Constraints

$\prec_e$: set of events that happen before ***e***

W_v^x : set of writes that write value ***v*** to a variable, ***x***

W^x : set of writes that write other values to ***x***

$$\Phi_{validity} = \bigwedge_{r \in \prec_e} \Phi_{value}(r, v),$$

$\phi_{value}(r, v)$ enforces **r** returns the value **v**

$$\Phi_{value}(r, v) \equiv \bigvee_{w \in W_v^x} (\Phi_{validity}(w) \wedge O_w < O_r$$
$$\bigwedge_{w \neq w' \in W^x} (O_{w'} < O_w \vee O_r < O_{w'}))$$

- every read ***r*** before ***e***, return the same value ***v***
- match ***r*** to a write that writes the value ***v*** to the same location

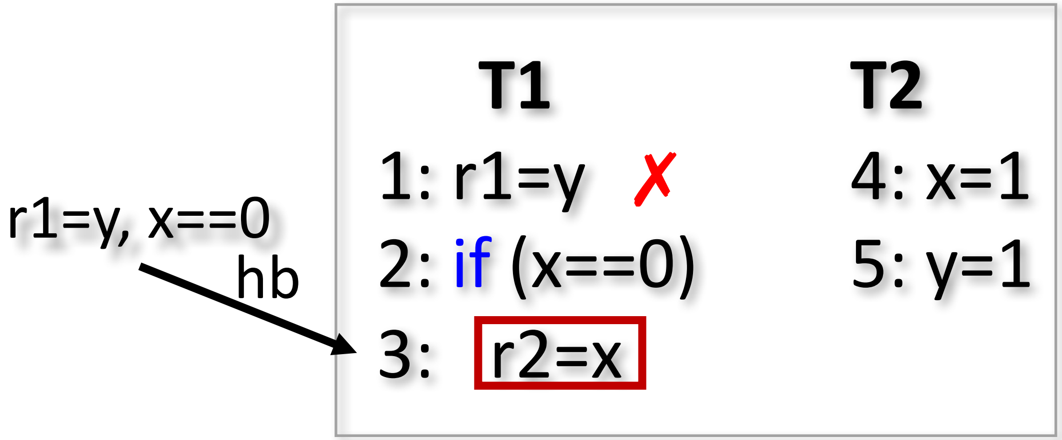
Limitations

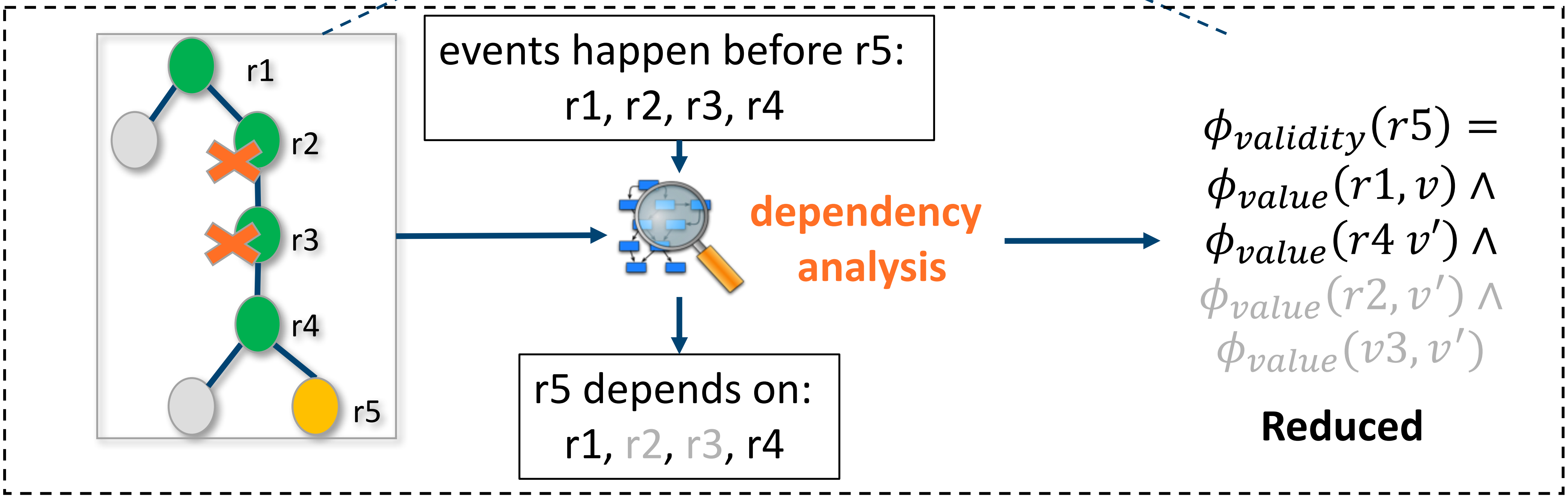
Most events are reads and writes in a trace

- Complicated constraints, **cubic** in the size of the trace

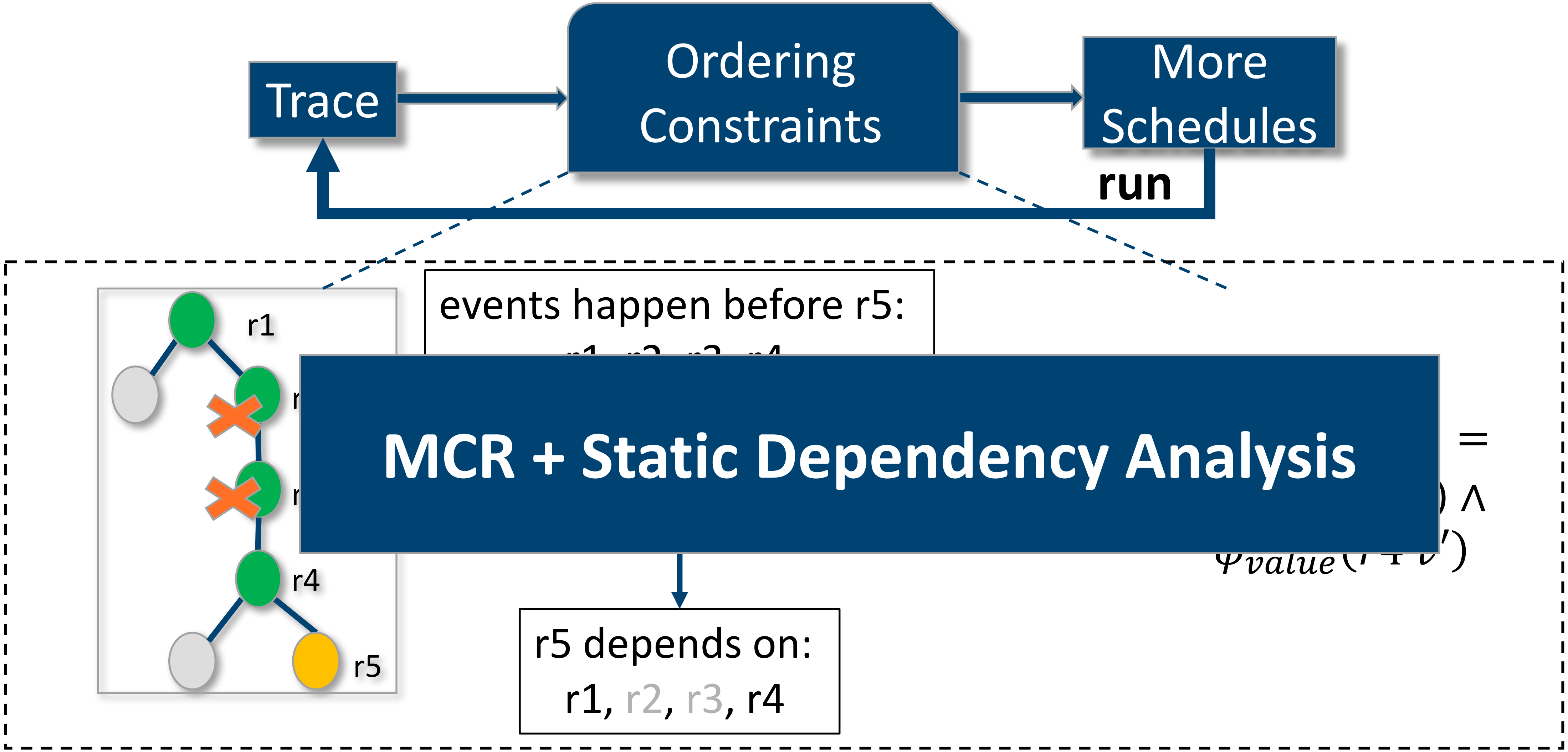
Just a few reads influence the reachability of a later event

- Construct unnecessary constraints



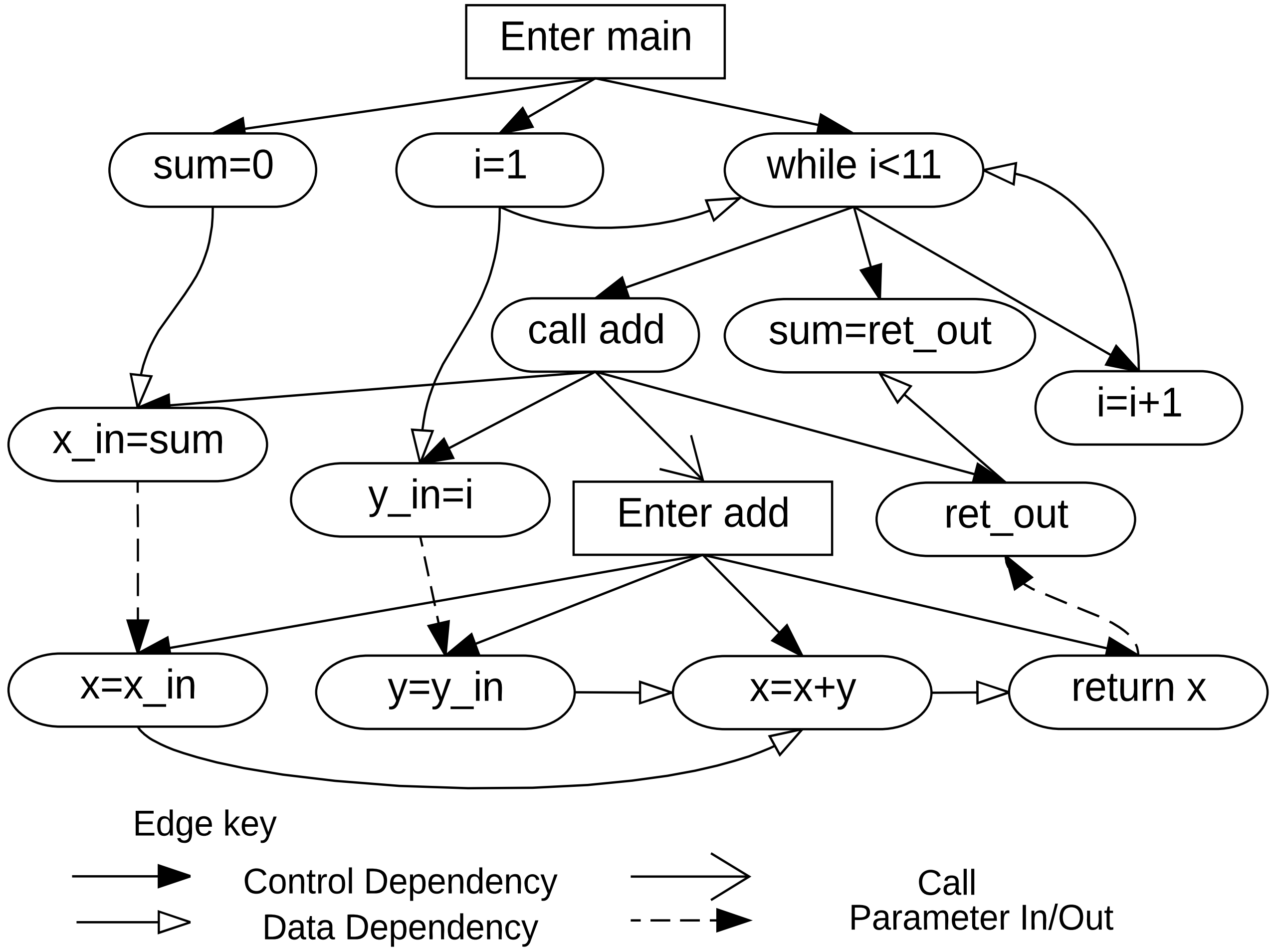

r1
r2
r3
r4
r5
events happen before r5:
r1, r2, r3, r4
dependency
analysis
r5 depends on:
r1, r2, r3, r4
$\phi_{validity}(r5) =$
$\phi_{value}(r1, v) \wedge$
$\phi_{value}(r4\ v') \wedge$
$\phi_{value}(r2, v') \wedge$
$\phi_{value}(v3, v')$
Reduced

Our Approach

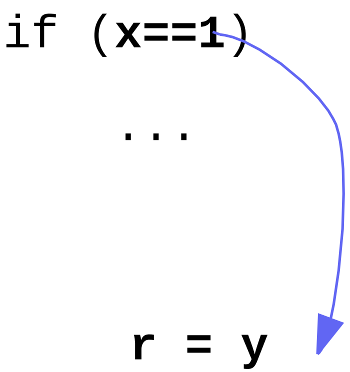


System Dependency Graph (SDG)

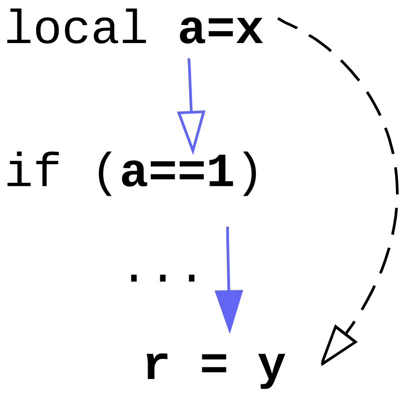
```
Procedure main()
   sum = 0;
   i = 1;
   while i<11:
       sum = add(sum, i);
       i = i+1;


Procedure add(x,y)
   x = x+y;
   return x;
```



Control Dependency



(a)



(b)

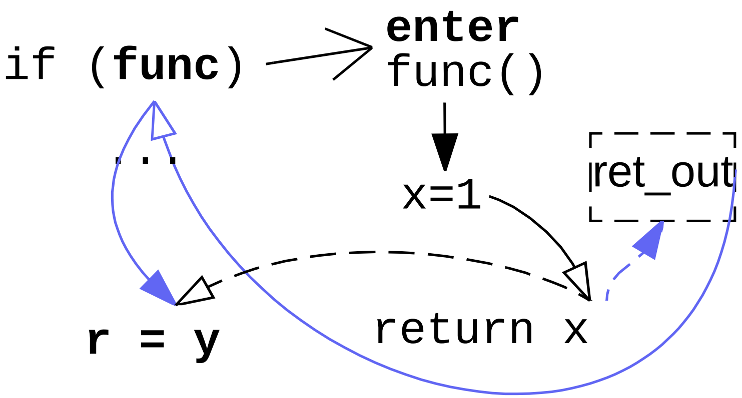
Case a: an event is directly depends on a read operation evaluated by an if predicate

$$x == 1 \xrightarrow{DD \cdot CD} r = y$$

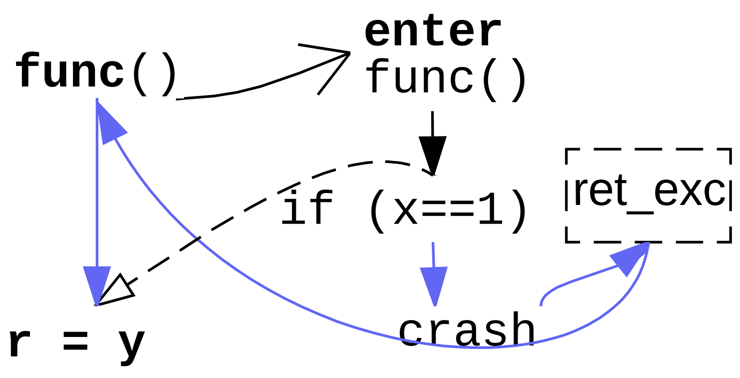
Case b: the dependency may be transmitted via a data dependency

$$a = x \xrightarrow{DD \cdot DD \cdot CD} r = y$$

Control Dependency



(c)



(d)

Case c: the evaluation may depend on the return value of another procedure

$$return\ x \xrightarrow{PO \cdot DD \cdot DD \cdot CD} r = y$$

Case d: the read may depend on a if predicate in a different procedure

$$x == 1 \xrightarrow{CD \cdot CD \cdot CD \cdot CD} r = y$$

Control Dependency

Definition: given two nodes n1 and n2 in an SDG, we use n1 δ^cn2 to denote that n2 is control dependent on n1

$$n1\ \delta^c\ n2 \Leftrightarrow n1 \xrightarrow{e^*CD} n2,$$
$$e \ := \ null$$
$$|CD\ |DD\ |PI\ |PO\ |CL$$

CD: control dependency
DD: data dependency
PI/O: parameter in/out
CL: call

Constraints Reduction

Main Idea:

Only enforce reads that are control-dependency related to return the same value

$\prec_\tau (e) \leftarrow$ **Happens-before**(τ, e)

$\prec_\tau^D (e) \leftarrow$ **DependencyComputation**$(\prec_\tau (e), e)$

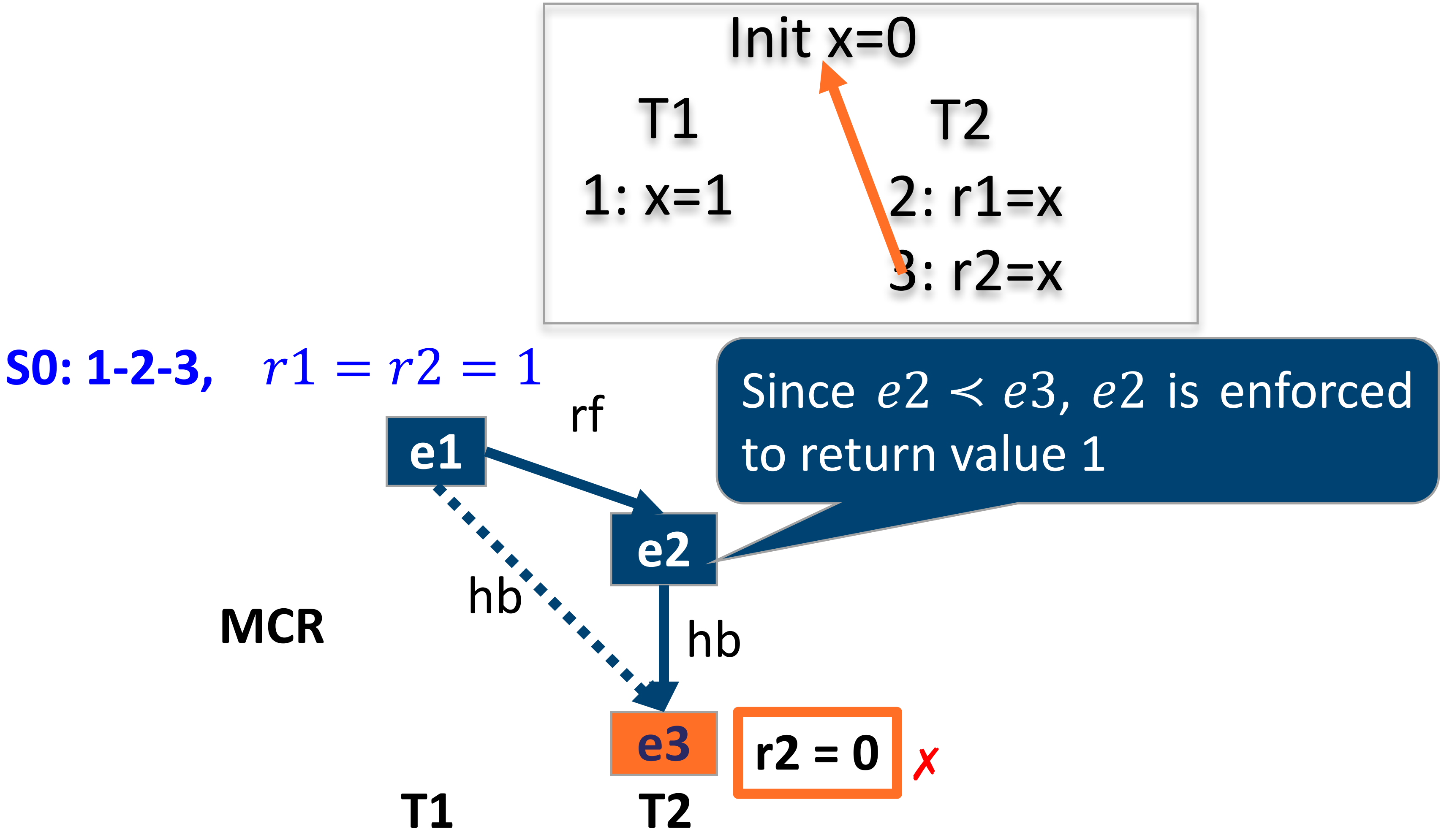
foreach *read* $r \in \prec_\tau^D (e)$ *with value* v **do**

// $\Phi_{value}(r, v)$ recursively call *DataValidityConstraints*()

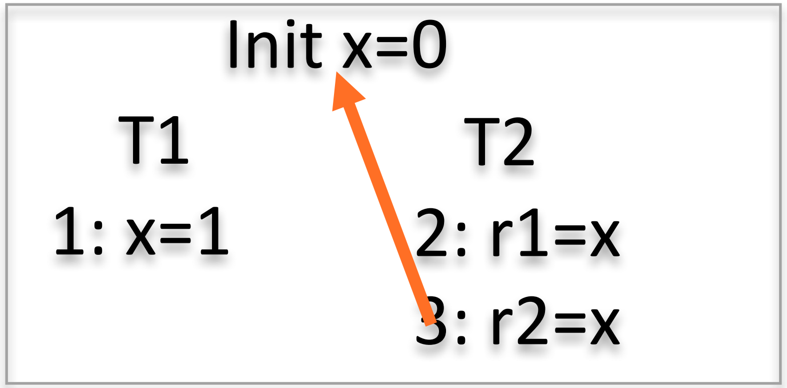
$\Phi_{validity} \wedge = \Phi_{value}(r, v)$

end

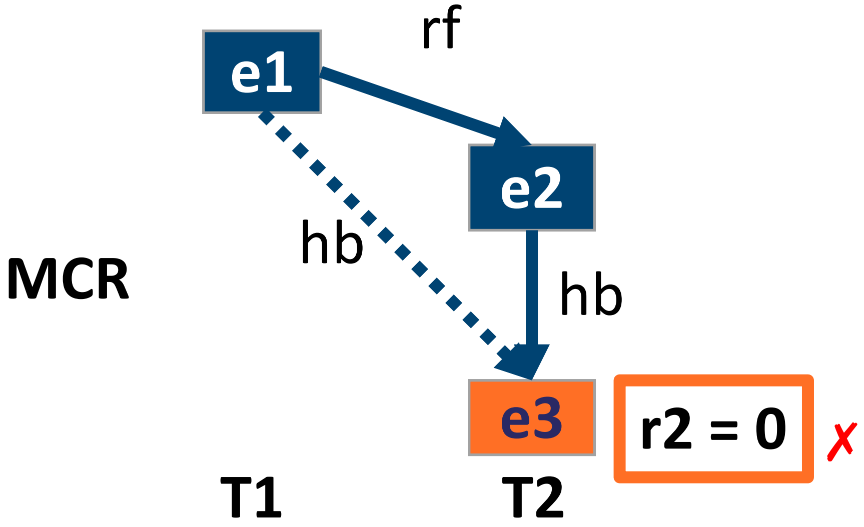
Redundancy Problem

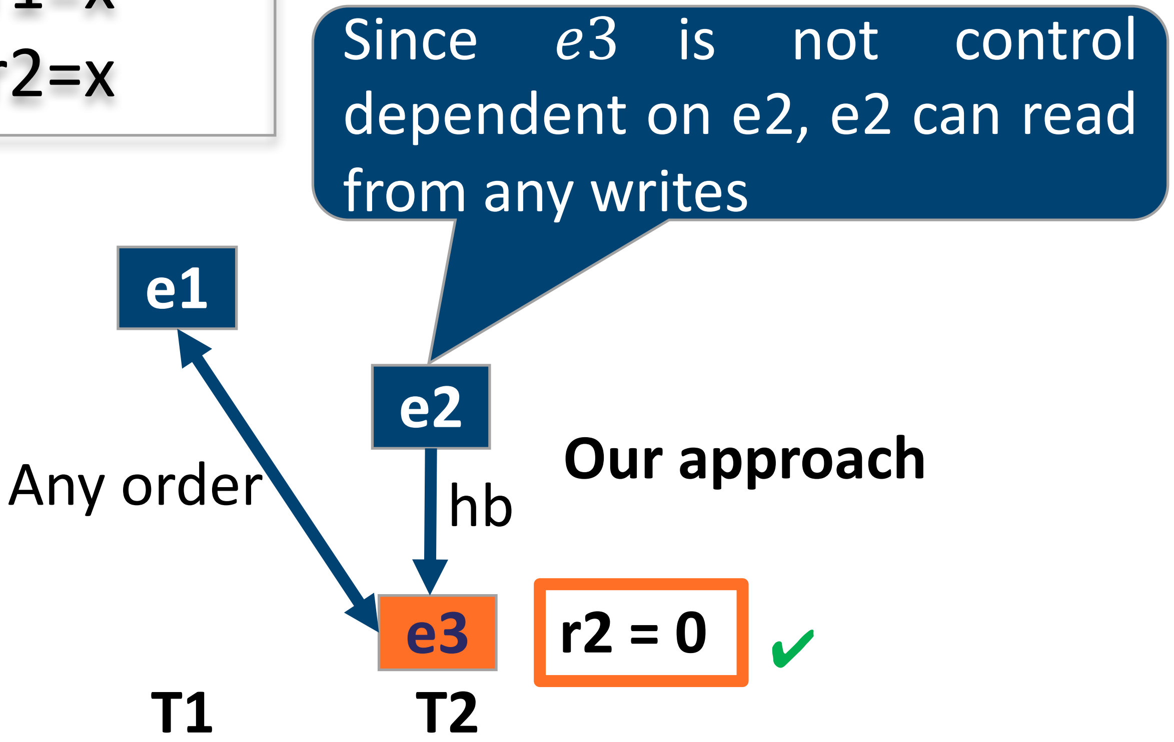


Redundancy Problem



S0: 1-2-3, $r1 = r2 = 1$





Solution to Redundancy Problem

We treat the events into two categories:

1. target read: a read considered to see a different value
2. other events

$\prec_\tau (e) \leftarrow$ **Happens-before**(τ, e)

```
// target read:  read considered to return new values
```

if *e is not a* TARGET READ **then**

  $\prec_\tau^D (e) \leftarrow$ **DependencyComputation**$(\prec_\tau (e), e)$

end

foreach *read* $r \in \prec_\tau^D (e)$ *with value* v **do**

  // $\Phi_{value}(r, v)$ recursively call *DataValidityConstraints*()

  $\Phi_{validity} \wedge = \Phi_{value}(r, v)$

end

Evaluation

- Dependency analysis using JOANA[1] [Graf] and WALA[2]
- Comparisons with MCR
  - #reads/constraints reduced
  - solving time reduced
- Benchmarks [Huang, PLDI'15]

1. Joana: http://pp.ipd.kit.edu/projects/joana/
2. Wala: http://wala.sourceforge.net/wiki/index.php/Main_Page

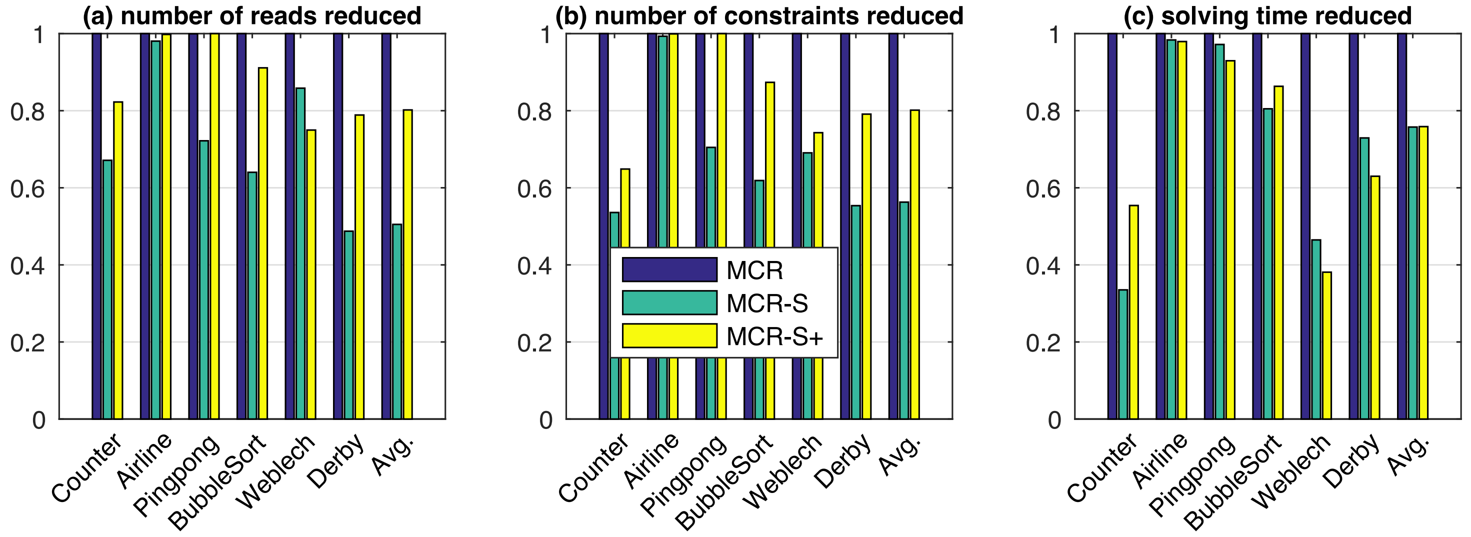
Benchmarks and SDG

Program	time(s)	memory(M)	#nodes	#edges
Counter	2.00	69	289	1,440
Airline	2.10	79	809	4,902
Pingpong	2.52	83	914	5,244
BubbleSort	2.14	81	911	5,710
Pool	3.67	75	2,848	17,586
StringBuf	2.96	111	2,129	12,310
Weblech	8.01	219	22,094	167,492
Derby	69.67	1,385	115,658	2,409,784

	time	memory
Avg.	11.6s	263M

Comparison with MCR

- MCR-S: Optimization with redundant executions
- MCR-S+: No redundancy, but less reads reduced



Approach	MCR-S	MCR-S+
Reads	27.1% ↓	12.1% ↓
Constraints	31.6% ↓	15.7% ↓
Solving time	27.8% ↓	26.2% ↓

Conclusion & Future Work

- Improvement over MCR
  - #reads/constraints: 12.1% - 27.1% , 15.7% - 31.6
  - solving time: ~27%
- Future work
  - take input non-determinism into consideration
  - release the tool

Thank You