Data-centric Dynamic Partial Order Reduction

Marek Chalupa Krishnendu Chatterjee **Andreas Pavlogiannis**
Nishant Sinha Kapil Vaidya

Concurrency Bugs

Thread 1: Withdraw(x)

if balance $\geqslant x$ **then**
    balance $\leftarrow$ balance $- x$

Concurrency Bugs

Thread 1: Withdraw(x)

if balance $\geqslant x$ **then**
    balance $\leftarrow$ balance $- x$

Thread 2: Withdraw(x)

if balance $\geqslant x$ **then**
    balance $\leftarrow$ balance $- x$

Concurrency Bugs

Thread 1: Withdraw(x)	**Thread 2:** Withdraw(x)
if balance $\geqslant x$ **then**	**if** balance $\geqslant x$ **then**
balance $\leftarrow$ balance $- x$	balance $\leftarrow$ balance $- x$

Withdraw(5)   Withdraw(5)

balance $= 8$

Concurrency Bugs

Thread 1: Withdraw(x)

```
if balance ⩾ x then
    balance ← balance − x
```

Thread 2: Withdraw(x)

```
if balance ⩾ x then
    balance ← balance − x
```

Withdraw(5)   Withdraw(5)

balance $= 8$

Concurrency Bugs

Thread 1: Withdraw(x)

if balance $\geqslant x$ **then**
    balance $\leftarrow$ balance $- x$

Thread 2: Withdraw(x)

if balance $\geqslant x$ **then**
    balance $\leftarrow$ balance $- x$

Withdraw(5)

Withdraw(5)

balance $= 8$

Concurrency Bugs

Thread 1: Withdraw(x)

if balance $\geqslant x$ **then**
    balance $\leftarrow$ balance $- x$

Thread 2: Withdraw(x)

if balance $\geqslant x$ **then**
    balance $\leftarrow$ balance $- x$

Withdraw(5)   Withdraw(5)

balance $= 8 \rightarrow 3$

Concurrency Bugs

Thread 1: Withdraw(x)

if balance $\geqslant x$ **then**
  balance $\leftarrow$ balance $- x$

Thread 2: Withdraw(x)

if balance $\geqslant x$ **then**
  balance $\leftarrow$ balance $- x$

Withdraw(5)

Withdraw(5)

$$\text{balance} = 8 \rightarrow 3 \rightarrow -2$$

Concurrency Bugs

Thread 1: Withdraw(x)
if balance $\geqslant x$ **then**
balance $\leftarrow$ balance $- x$

Thread 2: Withdraw(x)
if balance $\geqslant x$ **then**
balance $\leftarrow$ balance $- x$

Withdraw(5) Withdraw(5)

$$\text{balance} = 8 \rightarrow 3 \rightarrow -2$$

- No control over scheduling
- "Heisenbugs" lie in scheduling subtleties
- Formal verification to the rescue

Concurrency Setting

Concurrency Setting

- Deterministic processes
  - No randomization
  - Fixed inputs
- All nondeterministic behavior comes from the scheduler
- Goal local-state reachability: catch bugs, e.g. assertion violations

Concurrency Setting

- Deterministic processes
  - No randomization
  - Fixed inputs
- All nondeterministic behavior comes from the scheduler
- Goal local-state reachability: catch bugs, e.g. assertion violations

- Algorithmic problem: visit all local states of each process
- **Explicit state:** visiting each state must be fast
- **Stateless:** cannot remember all system states

Concurrency Setting

- Deterministic processes
  - No randomization
  - Fixed inputs
- All nondeterministic behavior comes from the scheduler
- Goal local-state reachability: catch bugs, e.g. assertion violations

- Algorithmic problem: visit all local states of each process
- **Explicit state:** visiting each state must be fast
- **Stateless:** cannot remember all system states

- Examine all traces
- $n!$ many
- We can do better: DPOR

Commutative Events

Definition

A pair of events (e_1, e_2) is **non-commutative** if

- e_1 and e_2 are in the same process, or
- e_1, e_2 use the same variable, and at last one is a write

The Mazurkiewicz Equivalence

Definition

Two traces t_1, t_2 are **Mazuriekwicz equivalent**, written $t_1 \sim_M t_2$, if

- $\text{Events}(t_1) = \text{Events}(t_2) = E$, and
- for every non-commutative pair $(e_1, e_2) \in E \times E$,

$$e_1 \rightarrow_{t_1} e_2 \qquad \text{iff}$$

$$e_1 \rightarrow_{t_2} e_2$$

The Mazurkiewicz Equivalence

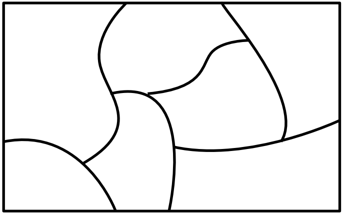
Definition

Two traces t_1, t_2 are **Mazuriekwicz equivalent**, written $t_1 \sim_M t_2$, if

- $\text{Events}(t_1) = \text{Events}(t_2) = E$, and
- for every non-commutative pair $(e_1, e_2) \in E \times E$,

$$e_1 \rightarrow_{t_1} e_2 \qquad \text{iff}$$
$$e_1 \rightarrow_{t_2} e_2$$

T



$$t_1 \sim_M t_2 \implies \text{local states agree}$$

$$n! \mapsto |\mathcal{T}/ \sim_M|$$

Focus on 2 processes

Motivating Example

Process p_1 :	Process p_2 :
$w_x^1 \quad r_x^1$	$w_x^2 \quad r_x^2$

Motivating Example

Process p_1 :	Process p_2 :
$w_x^1 \quad r_x^1$	$w_x^2 \quad r_x^2$

$\mathbf{t_1}$

w_x^1

r_x^1

w_x^2

r_x^2

Motivating Example

Process p_1 :	Process p_2 :
$w_x^1 \quad r_x^1$	$w_x^2 \quad r_x^2$

$\mathbf{t_1}$	$\mathbf{t_2}$
w_x^1	w_x^2
r_x^1	r_x^2
w_x^2	w_x^1
r_x^2	r_x^1

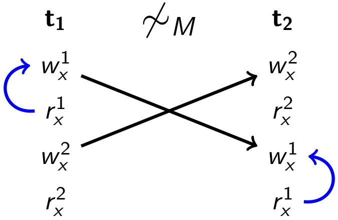
Motivating Example

Process p_1 :	Process p_2 :
$w_x^1 \quad r_x^1$	$w_x^2 \quad r_x^2$

$\mathbf{t_1}$	$\not\sim_M$	$\mathbf{t_2}$
w_x^1		w_x^2
r_x^1		r_x^2
w_x^2		w_x^1
r_x^2		r_x^1

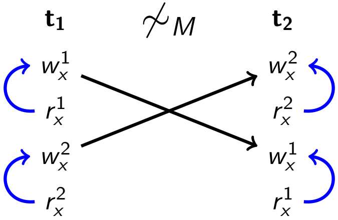
Motivating Example

Process p_1 :	Process p_2 :
$w_x^1 \quad r_x^1$	$w_x^2 \quad r_x^2$



Motivating Example

Process p_1 :	Process p_2 :
$w_x^1 \quad r_x^1$	$w_x^2 \quad r_x^2$



The Observation Equivalence

Definition

Observation O_t : Reads$(t) \rightarrow$ Writes(t)

Definition

Two traces t_1, t_2 are **Observation equivalent**, written $t_1 \sim_O t_2$, if

- Events(t_1) = Events$(t_2) = E$, and
- for each read $r \in E$,

$$O_{t_1}(r) = O_{t_2}(r)$$

The Observation Equivalence

Definition

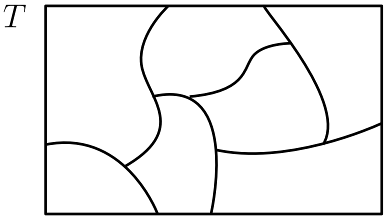
Observation O_t : Reads$(t) \to$ Writes(t)

Definition

Two traces t_1, t_2 are **Observation equivalent**, written $t_1 \sim_O t_2$, if

- Events(t_1) = Events$(t_2) = E$, and
- for each read $r \in E$,

$$O_{t_1}(r) = O_{t_2}(r)$$



$t_1 \sim_O t_2 \implies$ local states agree

Main Result

Theorem (1)

$\sim_M$ *refines* $\sim_O$.

Main Result

Theorem (1)

$\sim_M$ refines $\sim_O$.

Theorem (2)

$\sim_O$ can be exponentially coarser than $\sim_M$.

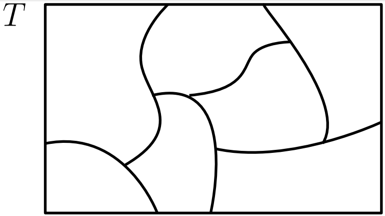
Main Result

Theorem (1)

$\sim_M$ *refines* $\sim_O$.

Theorem (2)

$\sim_O$ *can be exponentially coarser than* $\sim_M$.



Main Result

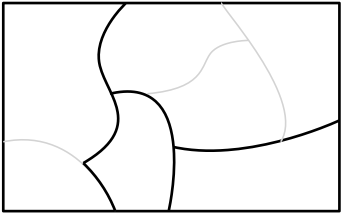
Theorem (1)

$\sim_M$ refines $\sim_O$.

Theorem (2)

$\sim_O$ can be exponentially coarser than $\sim_M$.

T



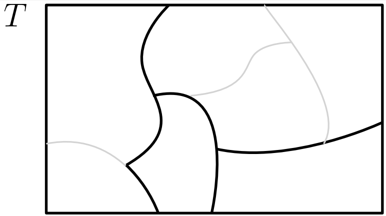
Main Result

Theorem (1)

$\sim_M$ refines $\sim_O$.

Theorem (2)

$\sim_O$ can be exponentially coarser than $\sim_M$.



Theorem (3)

There exists an algorithm that explores every class of $\sim_O$

- *exactly once (optimal),*
- *while spending polynomial time per class*

Theorem 1: $\sim_M$ refines $\sim_O$

- Consider traces t_1, t_2 with $t_1 \not\sim_O t_2$

Theorem 1: $\sim_M$ refines $\sim_O$

- Consider traces t_1, t_2 with $t_1 \not\sim_O t_2$

Theorem 1: $\sim_M$ refines $\sim_O$

- Consider traces t_1, t_2 with $t_1 \not\sim_O t_2$

Theorem 1: $\sim_M$ refines $\sim_O$

- Consider traces t_1, t_2 with $t_1 \not\sim_O t_2$

- Consider traces t_1, t_2 with $t_1 \not\sim_O t_2$

Theorem 1: $\sim_M$ refines $\sim_O$

- Consider traces t_1, t_2 with $t_1 \not\sim_O t_2$

Theorem 1: $\sim_M$ refines $\sim_O$

- Consider traces t_1, t_2 with $t_1 \not\sim_O t_2$

- In all cases, $t_1 \not\sim_M t_2$

Theorem 2: $\sim_O$ exponentially coarser than $\sim_M$

Process p_1 :	Process p_2 :
1. write x	1. write x
2. write x	2. write x
...	...
$n+1$. read x	$n+1$. read x

Theorem 2: $\sim_O$ exponentially coarser than $\sim_M$

Process p_1 :	Process p_2 :
1. write x	1. write x
2. write x	2. write x
...	...
$n+1$. read x	$n+1$. read x

$$|\mathcal{T}/\sim_O| = O(n) \qquad |\mathcal{T}/\sim_M| = \Omega(2^n)$$

Theorem 3: Exists optimal, fast algorithm

Theorem 3: Exists optimal, fast algorithm

A bit more involved. . .

Realizing observation functions

Algorithmic problem

Given observation function $\mathcal{O} : \text{Reads} \mapsto \text{Writes}$

- Construct trace t with $\mathcal{O}_t = \mathcal{O}$, or
- Return False if $\mathcal{O}$ is unrealizable

Realizing observation functions

Algorithmic problem

Given observation function $\mathcal{O}$: Reads $\mapsto$ Writes

- Construct trace t with $\mathcal{O}_t = \mathcal{O}$, or
- Return False if $\mathcal{O}$ is unrealizable

- Can construct $\mathcal{O}$ encoding assertion violations
- Not easier than our original problem

Realizing well-formed observation functions (1)

Well-formed observation functions

An observation function $\mathcal{O}$ is well-formed if . . .

Realizing well-formed observation functions (1)

Well-formed observation functions

An observation function $\mathcal{O}$ is well-formed if ...

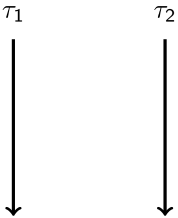
- Defines a local execution in each process

Realizing well-formed observation functions (1)

Well-formed observation functions

An observation function $\mathcal{O}$ is well-formed if ...

- Defines a local execution in each process

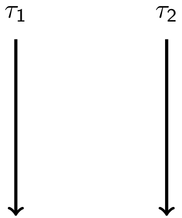


Realizing well-formed observation functions (1)

Well-formed observation functions

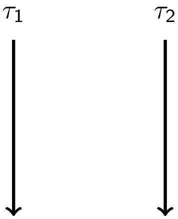
An observation function $\mathcal{O}$ is well-formed if ...

- Defines a local execution in each process

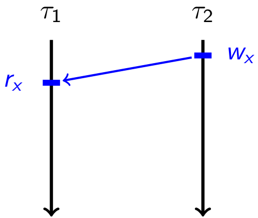


- Any t that realizes $\mathcal{O}$ must be a linearization of $\tau_1||\tau_2$

Realizing well-formed observation functions (2)

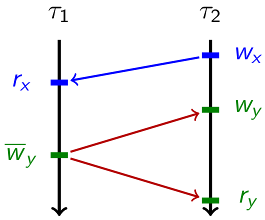


Realizing well-formed observation functions (2)



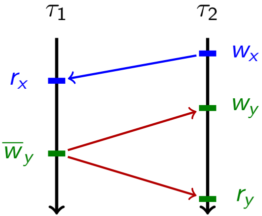
- Observation constraints $\rightarrow$

Realizing well-formed observation functions (2)



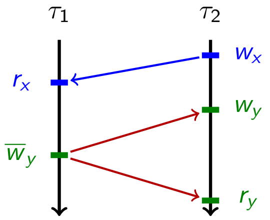
- Observation constraints $\rightarrow$
- Sequential consistency constraints $\rightarrow$
  - **2 SAT**: If $\overline{w}_y \rightarrow r_y$ then $\overline{w}_y \rightarrow w_y$

Realizing well-formed observation functions (2)



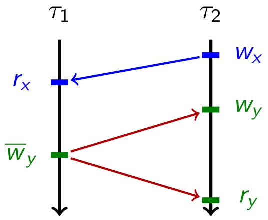
- Observation constraints $\rightarrow$
- Sequential consistency constraints $\rightarrow$
  - **2 SAT**: If $\overline{w}_y \rightarrow r_y$ then $\overline{w}_y \rightarrow w_y$
- Transitivity constraints
  - If $(e_1 \rightarrow e_2$ and $e_2 \rightarrow e_3)$ then $e_1 \rightarrow e_3$
  - 3 SAT ☹

Realizing well-formed observation functions (2)



- Observation constraints $\rightarrow$
- Sequential consistency constraints $\rightarrow$
  - **2 SAT**: If $\overline{w}_y \rightarrow r_y$ then $\overline{w}_y \rightarrow w_y$
- Transitivity constraints
  - If $(e_1 \rightarrow e_2$ and $e_2 \rightarrow e_3)$ then $e_1 \rightarrow e_3$
  - 3 SAT ☹
  - For 2 processes, every triplet of events has an ordered pair!
  - If (~~$e_1 \rightarrow e_2$ and~~ $e_2 \rightarrow e_3$) then $e_1 \rightarrow e_3$
  - If ($e_1 \rightarrow e_2$ ~~and $e_2 \rightarrow e_3$~~) then $e_1 \rightarrow e_3$

Realizing well-formed observation functions (2)



- Observation constraints $\rightarrow$
- Sequential consistency constraints $\rightarrow$
  - **2 SAT**: If $\overline{w}_y \rightarrow r_y$ then $\overline{w}_y \rightarrow w_y$
- Transitivity constraints
  - If ($e_1 \rightarrow e_2$ and $e_2 \rightarrow e_3$) then $e_1 \rightarrow e_3$
  - 3 SAT ☹
  - For 2 processes, every triplet of events has an ordered pair!
  - If (~~$e_1 \rightarrow e_2$ and~~ $e_2 \rightarrow e_3$) then $e_1 \rightarrow e_3$
  - If ($e_1 \rightarrow e_2$ ~~and $e_2 \rightarrow e_3$~~) then $e_1 \rightarrow e_3$
  - 2 SAT ☺

Theorem

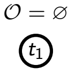
For 2 processes, realizing a well-formed observation requires polynomial time.

Data-centric DPOR

1. Start with an empty observation
2. While at node u with observation $\mathcal{O}_u$

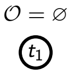
$$\mathcal{O} = \varnothing$$

1. Start with an empty observation
2. While at node u with observation $\mathcal{O}_u$
   1. Get a (any!) trace t_u realizing $\mathcal{O}_u$



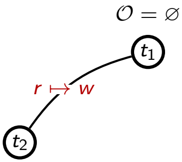
Data-centric DPOR

1. Start with an empty observation
2. While at node u with observation $\mathcal{O}_u$
   1. Get a (any!) trace t_u realizing $\mathcal{O}_u$
   2. Mutation: take every
      1. $r \in \text{Events}(t_u) \setminus \mathcal{O}_u$, and
      2. $w \in \text{Events}(t_u)$ with $\text{Confl}(r, w)$



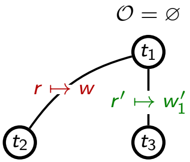
Data-centric DPOR

1. Start with an empty observation
2. While at node u with observation $\mathcal{O}_u$
    1. Get a (any!) trace t_u realizing $\mathcal{O}_u$
    2. Mutation: take every
        1. $r \in \text{Events}(t_u) \setminus \mathcal{O}_u$, and
        2. $w \in \text{Events}(t_u)$ with $\text{Confl}(r, w)$
        3. Construct successor v of u with $\mathcal{O}_v = \mathcal{O}_u \cup \{r \mapsto w\}$
        4. (If $\mathcal{O}_v$ not well-formed, skip)



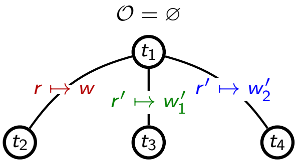
Data-centric DPOR

1. Start with an empty observation
2. While at node u with observation $\mathcal{O}_u$
   1. Get a (any!) trace t_u realizing $\mathcal{O}_u$
   2. Mutation: take every
      1. $r \in \text{Events}(t_u) \setminus \mathcal{O}_u$, and
      2. $w \in \text{Events}(t_u)$ with $\text{Confl}(r, w)$
      3. Construct successor v of u with $\mathcal{O}_v = \mathcal{O}_u \cup \{r \mapsto w\}$
      4. (If $\mathcal{O}_v$ not well-formed, skip)



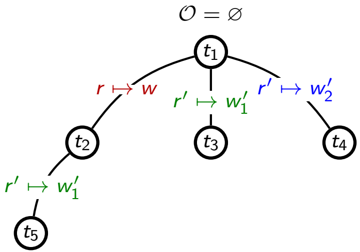
Data-centric DPOR

1. Start with an empty observation
2. While at node u with observation $\mathcal{O}_u$
   1. Get a (any!) trace t_u realizing $\mathcal{O}_u$
   2. Mutation: take every
      1. $r \in \text{Events}(t_u) \setminus \mathcal{O}_u$, and
      2. $w \in \text{Events}(t_u)$ with $\text{Confl}(r, w)$
      3. Construct successor v of u with $\mathcal{O}_v = \mathcal{O}_u \cup \{r \mapsto w\}$
      4. (If $\mathcal{O}_v$ not well-formed, skip)



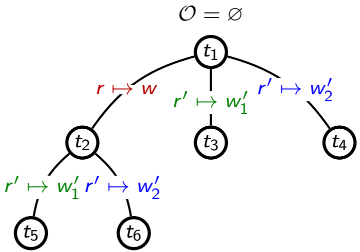
Data-centric DPOR

1. Start with an empty observation
2. While at node u with observation $\mathcal{O}_u$
   1. Get a (any!) trace t_u realizing $\mathcal{O}_u$
   2. Mutation: take every
      1. $r \in \text{Events}(t_u) \setminus \mathcal{O}_u$, and
      2. $w \in \text{Events}(t_u)$ with $\text{Confl}(r, w)$
      3. Construct successor v of u with $\mathcal{O}_v = \mathcal{O}_u \cup \{r \mapsto w\}$
      4. (If $\mathcal{O}_v$ not well-formed, skip)



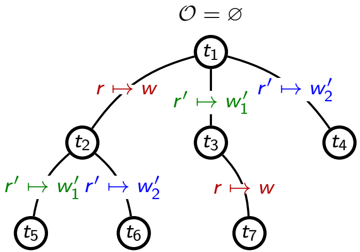
Data-centric DPOR

1. Start with an empty observation
2. While at node u with observation $\mathcal{O}_u$
   1. Get a (any!) trace t_u realizing $\mathcal{O}_u$
   2. Mutation: take every
      1. $r \in \text{Events}(t_u) \setminus \mathcal{O}_u$, and
      2. $w \in \text{Events}(t_u)$ with $\text{Confl}(r, w)$
      3. Construct successor v of u with $\mathcal{O}_v = \mathcal{O}_u \cup \{r \mapsto w\}$
      4. (If $\mathcal{O}_v$ not well-formed, skip)



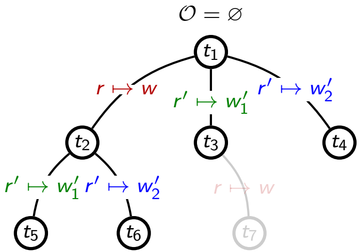
Data-centric DPOR

1. Start with an empty observation
2. While at node u with observation $\mathcal{O}_u$
   1. Get a (any!) trace t_u realizing $\mathcal{O}_u$
   2. Mutation: take every
      1. $r \in \text{Events}(t_u) \setminus \mathcal{O}_u$, and
      2. $w \in \text{Events}(t_u)$ with $\text{Confl}(r, w)$
      3. Construct successor v of u with $\mathcal{O}_v = \mathcal{O}_u \cup \{r \mapsto w\}$
      4. (If $\mathcal{O}_v$ not well-formed, skip)



Data-centric DPOR

1. Start with an empty observation
2. While at node u with observation $\mathcal{O}_u$
   1. Get a (any!) trace t_u realizing $\mathcal{O}_u$
   2. Mutation: take every
      1. $r \in \text{Events}(t_u) \setminus \mathcal{O}_u$, and
      2. $w \in \text{Events}(t_u)$ with $\text{Confl}(r, w)$
      3. Construct successor v of u with $\mathcal{O}_v = \mathcal{O}_u \cup \{r \mapsto w\}$
      4. (If $\mathcal{O}_v$ not well-formed, skip)
3. bookkeeping (local!) to guarantee optimality



Theorem

- *Every observation function visited once*
- *Total time* $O(\text{poly}(n) \cdot |\mathcal{T}/\sim_O|)$

Focus on $k \geqslant 2$ processes

$k \geqslant 2$ processes

Theorem

Realizing a well-formed observation function is NP-complete.

Hints on giving up either

- Polynomial time, or
- $\sim_O$ coarseness

Topologies

A graph $G = (V, E)$ depicts the communication topology

- $V = \{p_1, \dots, p_k\}$
- $(p_i, p_j) \in E$ iff processes p_i, p_j share a global variable

Acyclic topologies

- 2 processes, stars, pipelines, ...
- $\sim_O$ optimal
- Time $O\left(\text{poly}(n) \cdot |\mathcal{T}/\sim_O|\right)$

k processes

Acyclic topologies

- 2 processes, stars, pipelines, ...
- $\sim_O$ optimal
- Time $O\left(\text{poly}(n) \cdot |\mathcal{T}/\sim_O|\right)$

Arbitrary topologies

- Cliques, ...
- $\sim$ optimal
  - $\sim_M \leqslant \sim \leqslant \sim_O$
  - $\sim$ exponentially coarser than $\sim_M$
- Time $O\left(\text{poly}(n) \cdot |\mathcal{T}/\sim|\right)$

k processes

Acyclic topologies

- 2 processes, stars, pipelines, ...
- $\sim_O$ optimal
- Time $O\left(\text{poly}(n) \cdot |\mathcal{T}/ \sim_O|\right)$

Arbitrary topologies

- Cliques, ...
- $\sim$ optimal
  - $\sim_M \leqslant \sim \leqslant \sim_O$
  - $\sim$ exponentially coarser than $\sim_M$
- Time $O\left(\text{poly}(n) \cdot |\mathcal{T}/ \sim|\right)$

Space usage $O(n^3)$

Implementation & Experiments

- Implemented DC-DPOR for handling programins in C/pthreads
- Based on Nidhugg
- Conducted some experiments, comparing with Source-DPOR

Synthetic Benchmarks

```
// ---- Process 0 < j < 2 ----
i ← 0
while i < n do
    i ← i + 1
    last_id ← j
    x ← get_message(j)
    if last_id = j then
        return
end
```

```
// ---- Process 0 < j < 2 ----
i ← 0
while i < n do
    i ← i + 1
    last_id ← j
    x ← get_message(j)
    if last_id = j then
        return
end
```

Benchmark	Traces		Time (s)	
	DC-DPOR	S-DPOR	DC-DPOR	S-DPOR
opt_lock(12)	141	785,674	0.35	252.64
opt_lock(13)	153	2,056,918	0.36	703.90
opt_lock(14)	165	5,385,078	0.43	1,880.12
opt_lock(15)	177	-	0.46	-
opt_lock(50)	597	-	5.91	-
opt_lock(100)	1,197	-	43.82	-
opt_lock(200)	2,397	-	450.99	-

Benchmarks from SV-Comp (1)

Benchmark	Traces		Time (s)	
	DC-DPOR	S-DPOR	DC-DPOR	S-DPOR
fib_bench(4)	1,233	19,605	0.93	3.03
fib_bench(5)	8,897	218,243	7.41	37.82
fib_bench(6)	70,765	2,364,418	85.71	463.52

Benchmark	Traces		Time (s)	
	DC-DPOR	S-DPOR	DC-DPOR	S-DPOR
pthread_demo(8)	256	12,870	0.37	3.17
pthread_demo(10)	1,024	184,756	1.23	49.51
pthread_demo(12)	4,096	2,704,156	5.30	884.99

Benchmarks from SV-Comp (2)

Benchmark	Traces		Time (s)	
	DC-DPOR	S-DPOR	DC-DPOR	S-DPOR
parker(8)	1,254	3,343	1.52	1.33
parker(10)	2,411	6,212	5.03	3.96
parker(12)	4,132	10,361	8.09	5.62
parker(14)	6,529	16,022	11.96	6.86
parker(16)	9,714	23,427	19.89	10.85

- A new paradigm for DPOR
- Data-centric instead of Control-centric
- Coarser partitioning of the trace space
- Efficient exploration

Thank you!

Questions?

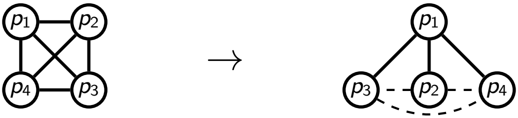
k processes

Acyclic topologies

- 2 processes, stars, pipelines, ...
- $\sim_O$ optimal
- Time $O\left(\text{poly}(n) \cdot |\mathcal{T}/\sim_O|\right)$

Arbitrary topologies

- Cliques, ...
- $\sim$ optimal
  - $\sim_M \leqslant \sim \leqslant \sim_O$
  - $\sim$ exponentially coarser than $\sim_M$
- Time $O\left(\text{poly}(n) \cdot |\mathcal{T}/\sim|\right)$



$\sim_O$ based
$\sim_M$ based

- Space usage $O(n^3)$
  - (compare with $\Omega(2^n)$ in Optimal Mazurkiewicz-based DPOR)

Data-centric DPOR

Lemma

- *Target observation function* $\mathcal{O}$*, and* t^* *a witness trace, i.e.,* $\mathcal{O}_{t^*} = \mathcal{O}$
- *Take any trace* t *with* $t \not\sim_O t^*$

Data-centric DPOR

Lemma

- *Target observation function $\mathcal{O}$, and t^* a witness trace, i.e., $\mathcal{O}_{t^*} = \mathcal{O}$*
- *Take any trace t with $t \not\sim_O t^*$*

Then, there exists a **first read** $r \in \text{Events}(t^*)$ *such that:*

Data-centric DPOR

Lemma

- *Target observation function $\mathcal{O}$, and t^* a witness trace, i.e., $\mathcal{O}_{t^*} = \mathcal{O}$*
- *Take any trace t with $t \not\sim_O t^*$*

Then, there exists a **first read** $r \in \text{Events}(t^*)$ *such that:*

- $r \in \text{Events}(t)$,

Data-centric DPOR

Lemma

- *Target observation function* $\mathcal{O}$, *and* t^* *a witness trace, i.e.,* $\mathcal{O}_{t^*} = \mathcal{O}$
- *Take any trace* t *with* $t \not\sim_{O} t^*$

Then, there exists a **first read** $r \in \text{Events}(t^*)$ *such that:*

- $r \in \text{Events}(t)$,
- $\mathcal{O}_{t^*}(r) \in \text{Events}(t)$,

Data-centric DPOR

Lemma

- *Target observation function* $\mathcal{O}$, *and* t^* *a witness trace, i.e.,* $\mathcal{O}_{t^*} = \mathcal{O}$
- *Take any trace* t *with* $t \not\sim_O t^*$

Then, there exists a **first read** $r \in \text{Events}(t^*)$ *such that:*

- $r \in \text{Events}(t)$,
- $\mathcal{O}_{t^*}(r) \in \text{Events}(t)$,
- $\mathcal{O}_{t^*}(r) \neq \mathcal{O}_t(r)$

Optimizations (we have a few)

Optimizations (we have a few)

Cycle detection

- Unit propagation in realizing observation functions
- $(a \implies b) \wedge a$ implies b
- Strengthens the PO
- Early cycle detection avoids 2 SAT altogether

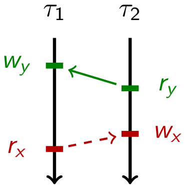
Optimizations (we have a few)

Cycle detection

- Unit propagation in realizing observation functions
- $(a \implies b) \wedge a$ implies b
- Strengthens the PO
- Early cycle detection avoids 2 SAT altogether

Burst mutations

- Standard algorithm accumulates mutations one-by-one
- Instead accumulate many at once

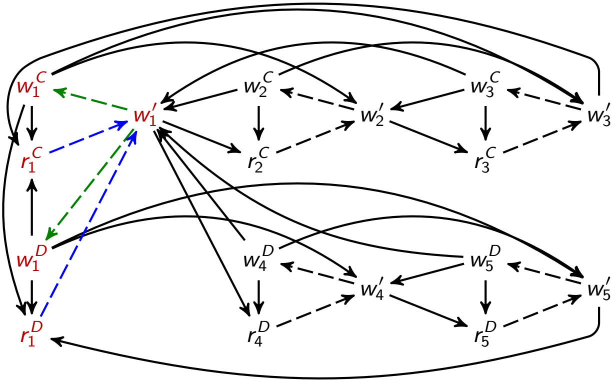


- Makes the recursion tree much shallower

Realizing Observation Functions is Hard

- Reduction from Monotone 1-in-3 SAT
- One unobserved w'_i for each variable x_i
- One observation $r_i^C \mapsto w_i C$ iff x_i appears in clause C
- Some extra happens-before edges

$$\phi = \underbrace{(x_1 \lor x_2 \lor x_3)}_{C} \land \underbrace{(x_1 \lor x_4 \lor x_5)}_{D}$$



Synthetic Benchmarks

```
// --------- Process j = 0 ---------
i ← n
while array[i] ≠ 0 do
    i ← i − 1
end

// --------- Process 1 < j ⩽ n ---------
array[j] ← array[j − 1] + 1
```

Synthetic Benchmarks

```
// --------- Process j = 0 ---------
i ← n
while array[i] ≠ 0 do
    i ← i − 1
end

// --------- Process 1 < j ⩽ n ---------
array[j] ← array[j − 1] + 1
```

Benchmark	Traces		Time (s)	
	DC-DPOR	S-DPOR	DC-DPOR	S-DPOR
lastzero(4)	38	2,118	0.21	0.84
lastzero(5)	113	53,172	0.34	19.29
lastzero(6)	316	1,765,876	0.63	856
lastzero(7)	937	-	1.8	-
lastzero(8)	3,151	-	9.32	-
lastzero(9)	12,190	-	47.97	-
lastzero(10)	52,841	-	383.12	-