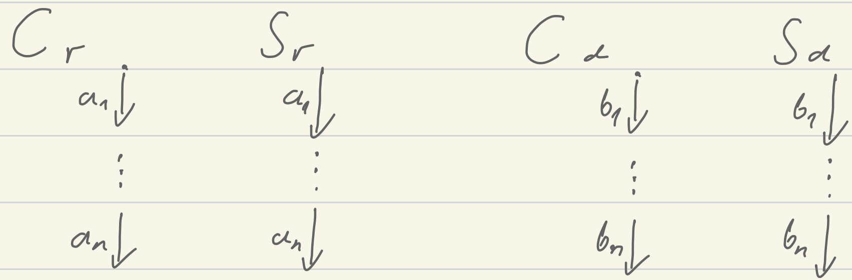
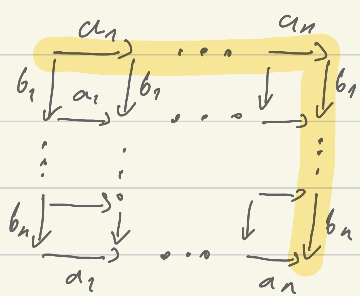
Revisiting partial-order reduction

Joint work with
Frédéric Herbreteau
Sarah Larroze-Jardine

What is POR





$a_1 \; I \; b_1$ as they use disjoint sets of processes

POR Literature

Beginnings

- "Stubborn sets for reduced space generation", Valmari, Petri Nets 1989
- "Using partial orders to improve automatic verification methods", Godefroid CAV'91
- "Verification of distributed programs using representative interleaving sequences", Katz, Peled, Distributed Computing'92
- "Stubborn Set Intuition Explained", Valmari, Hansen, Transactions on Petri Nets and Other Models of Concurrency, 2017

Dynamic POR (stateless)

- "Dynamic partial order reduction for model checkin software". Flanagan, Godefroid, POPL'05
- "Source Sets: A foundation for Optimal Dynamic Partial-Order Reduction", Abdulla, Aronis, Jonsson, Sagonas, POPL'14, JACM'17
- "Dynamic Partial Order Reduction", Marek Chalupa, Krishnendu Chatterjee, Andreas Pavlogiannis, Nishant Sinha, and Kapil Vaidya, POPL'17
- Chatterrjee CAV21 [chatterjee-por-cav21.pdf]
- "Truly Stateless, Optimal Dynamic Partial Order Reduction", Kokologiannakis, Marmanis, Gladstein, Vafeiadis, POPL'22

Back to state-full

- "Automated hypersafety verification" Farzan, Vandikas, CAV'19
- SymPaths: Symbolic Execution Meets Partial Order Reduction, de Boer, F.S., Bonsangue, M., Johnsen, E.B., Pun, V.K.I., Tapia Tarifa, S.L., Tveito, L. (2020)
- "Sound Sequentialization for Concurrent Program Verification", Farzan, Klumpp, Podelski, PLDI'22
- "A Pragmatic Approach to Stateful Partial Order Reduction", Ciriscі, Enea, Farzan, Mutluergil, VCAI'23
- "Stratified Commutativity in Verification Algorithms for Concurrent Progra
  Farzan, Klumpp, Podelski, POPL'23

Handling blocking (stateless)

- Awaiting for Godot: Stateless Model Checking that Avoids Executions where Nothing Happens, Jonsson, Lang, Agonas, FMCAD22
- Unblocking Dynamic Partial Order Reduction, Michalis Kokologiannakis , Iason Marmanis, and Viktor Vafeiadis, CAV'23

Where POR is useful

Model-checking programs (especially stateless model-checking)

“Dynamic partial order reduction for model checkin software”. Flanagan, Godefroid, POPL’05

Proving correctness of concurrent programs

“Sound Sequentialization for Concurrent Program Verification”, Farzan, Klumpp, Podelski, PLDI’22

Symbolic execution

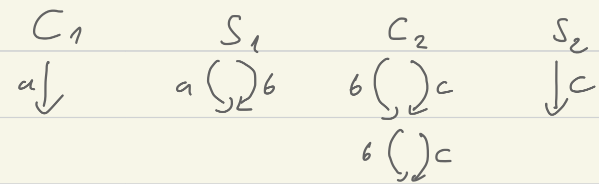
SymPaths: Symbolic Execution Meets Partial Order Reduction, de Boer, F.S., Bonsangue, M., Johnsen, E.B., Pun, V.K.I., Tapia Tarifa, S.L., Tveito, L. (2020)

Verification of timed systems

Probabilistic systems

How POR works

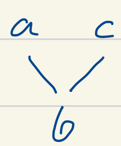
- Client / server programs

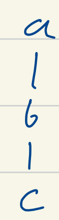


- Programs without cycles : initial and final state, complete run

- Trace equivalence : permuting independent actions

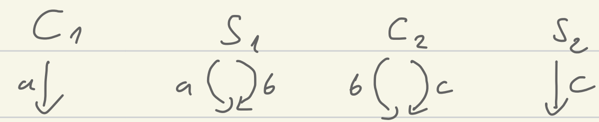
$acb \sim cab \qquad acb \not\sim abc$





How POR works

- Client / server programs



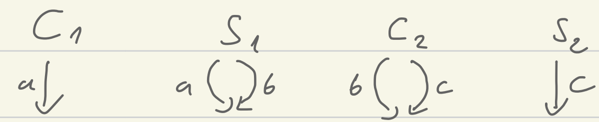
- Programs without cycles : initial and final state, complete run
- Trace equivalence : permuting independent actions $ac \sim ca$
- Goal : sound and complete transition system for a program

every complete path is a complete run

every complete run is trace equivalent to some complete path

How POR works

- Client / server programs



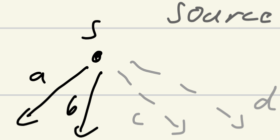
- Programs without cycles : initial and final state, complete run

- Trace equivalence : permuting independent actions $ac \sim ca$

- Goal : small sound and complete transition system for a program

- Source set for every node

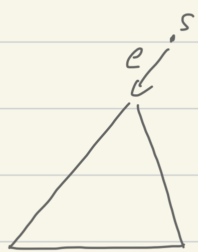
source$(s) = \{a, b\}$



Optimal on-the-fly POR

Constructs a tree of runs : every path is a complete run, no two paths are trace equiv

every run of the program is trace equi to some path



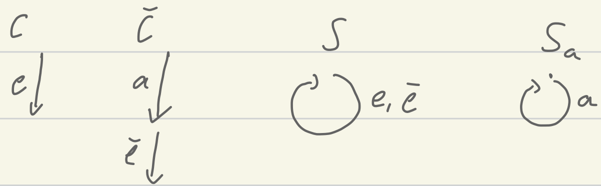
$e \in source(s)$

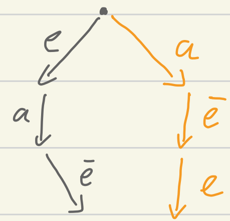
look for some patterns inside

eventually add some actions to source(s)

- "Source Sets: A foundation for Optimal Dynamic Partial-Order Reduction", Abdulla, Aronis, Jonsson, Sagonas, POPL'14, JACM'17

Race reversal



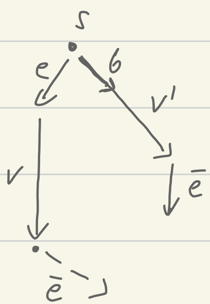


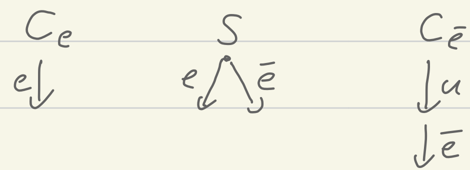
Optimal on-the-fly POR

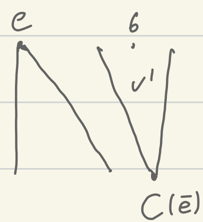
Constructs a tree of runs : every path is a complete run, no two paths are trace equiv

every run of the program is trace equi to some path

Race reversal







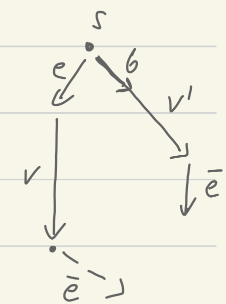
For such a situation add σ to source(s)

Optimal on-the-fly POR

- For optimality needed to keep $bv\bar{e} \in traces(s)$, and not $b \in source(s)$

  exponential memory

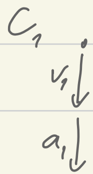
- Works only for non-blocking systems
- Improved to poly-memory
  - "Truly Stateless, Optimal Dynamic Partial Order Reduction", Kokologiannakis, Marmanis, Gladstein, Vafeiadis, POPL'22
- Blocking considered only very recently
- Lexicographic exploration is optimal too.

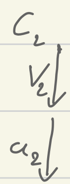


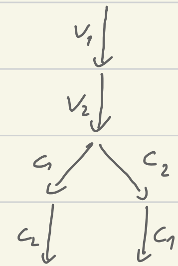
Handling blocking (stateless)

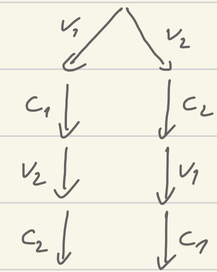
- Awaiting for Godot: Stateless Model Checking that Avoids Executions where Nothing Happens, Jonsson, Lang, Agonas, FMCAD22
- Unblocking Dynamic Partial Order Reduction, Michalis Kokologiannakis , Iason Marmanis, and Viktor Vafeiadis, CAV'23

How bad can optimal stateless get?

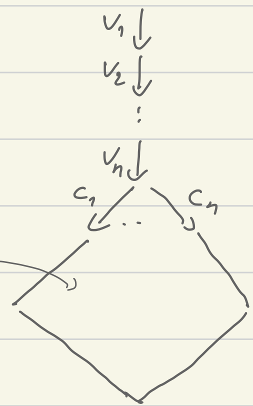






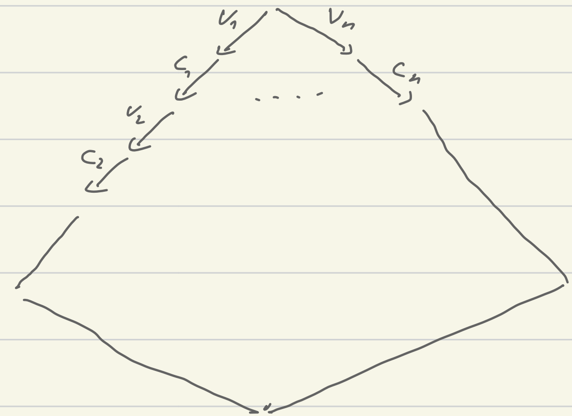


For n processes



all permutations of $c_1 .. c_n$

$n!$ paths

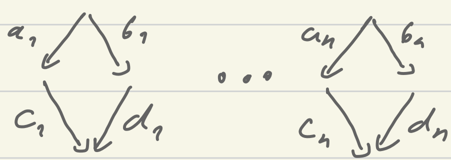


$n!$ paths

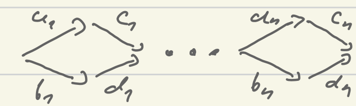
Every v_i repeated $>(n-1)!$ times

Statefull POR

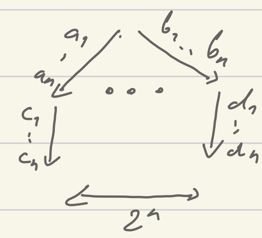
- Still based on persistent/ample sets + sleep sets
- Sensitive to exploration order



small graph:



big graph:



Both are optimal: every trace appears at most once
Stateless would produce the big graph

Good POR algorithm is impossible

$minTS(P)$: the smallest number of states of a sound and complete TS for P

Alg is <u>good</u> if given P constructs a sound and complete TS for P
of size $< q(minTS(P))$ time $< r(|P| + minTS(P))$

THM: If $P \neq NP$ then there is no good POR algorithm.

THM: If $P \neq NP$ then there is no good POR algorithm.

For a Boolean formula φ construct P_φ s.t

- If φ not SAT Then $minTS(P_\varphi) \leq 6|\varphi|$
- If φ SAT then $minTS(P_\varphi) > \#$ valuations satisfying φ

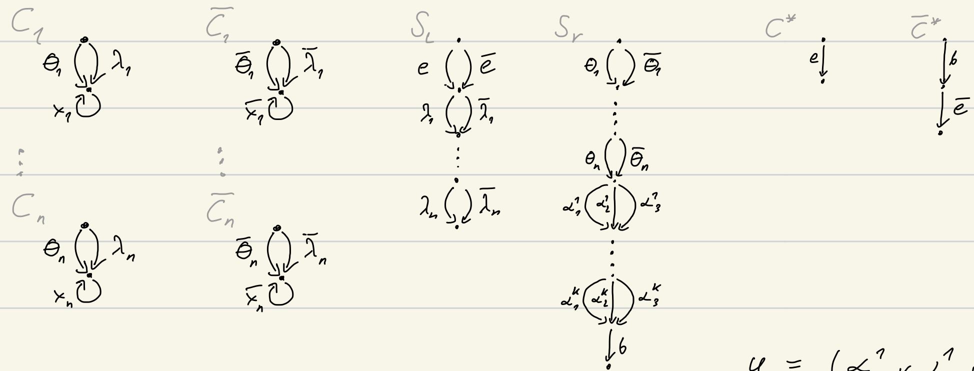
Proof: Take $\varphi \equiv \psi \wedge (z_1 \vee z_2) \wedge \ldots \wedge (z_{2m-1} \vee z_{2m})$

If ψ SAT Then φ has $> 2^m$ satisfying valuations

Run $Alg(P_\varphi)$ for $r(6|\varphi|)$ time.

If it stops ψ is not SAT

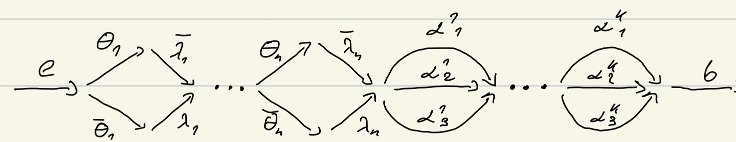
If it does not stop, ψ is SAT



$\varphi \equiv (\alpha_1^1 \vee \alpha_2^1 \vee \alpha_3^1) \wedge \dots \wedge (\alpha_1^k \vee \alpha_2^k \vee \alpha_3^k)$

α_j^i is x_l or $\bar{x}_l$ for some l

If φ notSAT then we have



If φ SAT then there are runs with $\bar{e}$ instead of e.

What do we have ?

Stateful POR algorithms that handle blocking
but are not good :)

CONCLUSIONS

We are interested in stateful POR methods

Finding subclasses for which good POR algorithms exist: acyclic architectures

Finding heuristics working in practice (based on reversals)

Impossibility results:

We cannot determine if a transition system will be small or large by simply looking at the program.
This means that there is no nice syntax for parallelism avoiding state explosion
(without limiting the kinds of models we can write in an important way).

Show that optimal stateless POR with blocking is impossible