Partial-order reduction

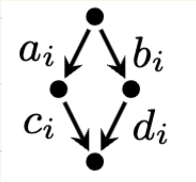
Lower bounds and heuristics

Igor Walukiewicz

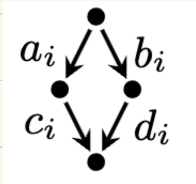
joint work with Frederic Herbreteau, Gerald Point, and Gautham Viswanathan

Partial-order reduction: example

P_1

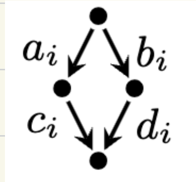


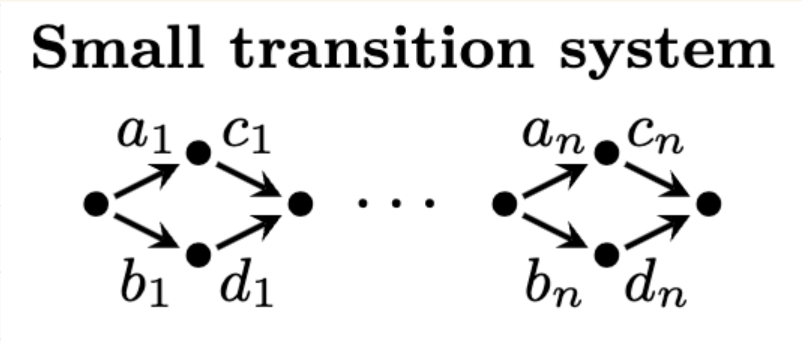
P_2

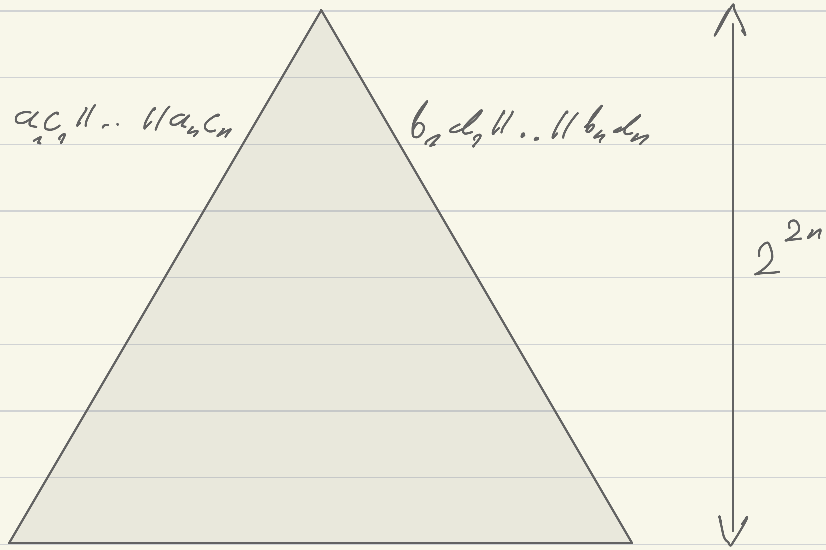


. . .

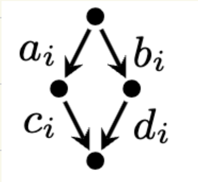
P_n

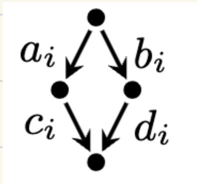




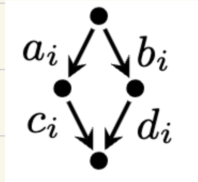


Partial-order reduction: example

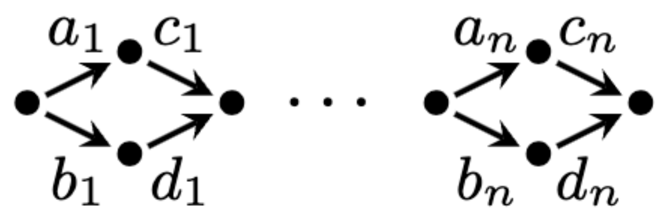
P_1 

P_2 

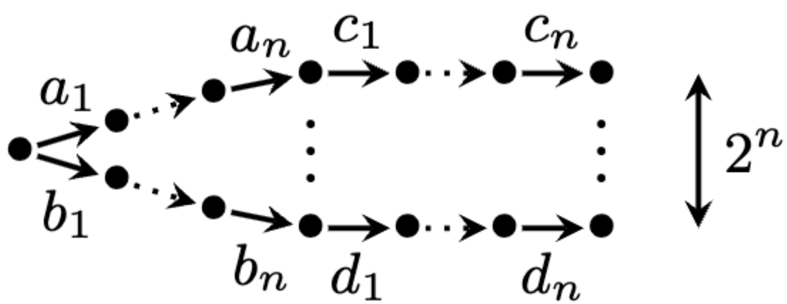
...

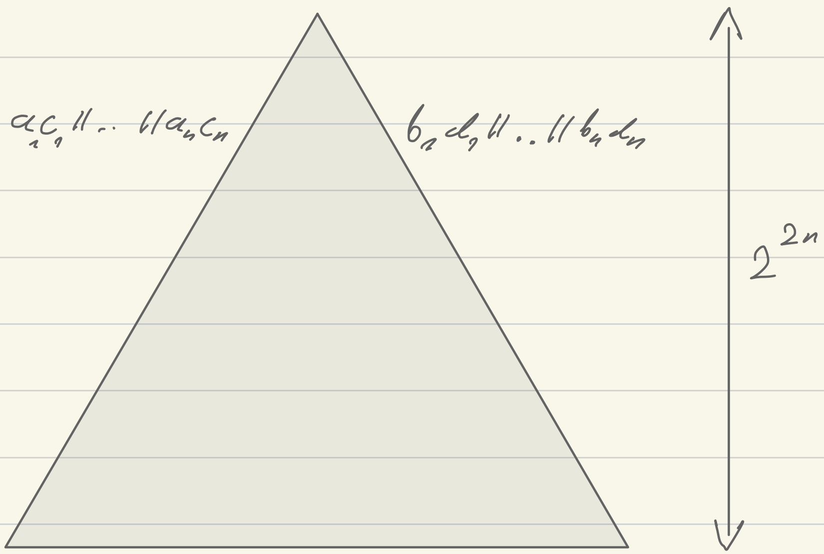
P_n 

Small transition system



Big transition system

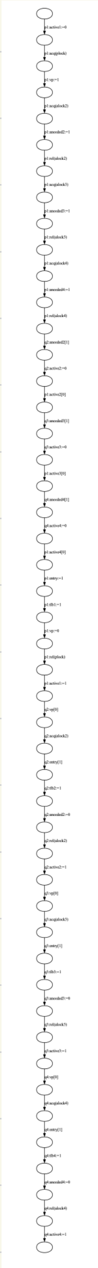




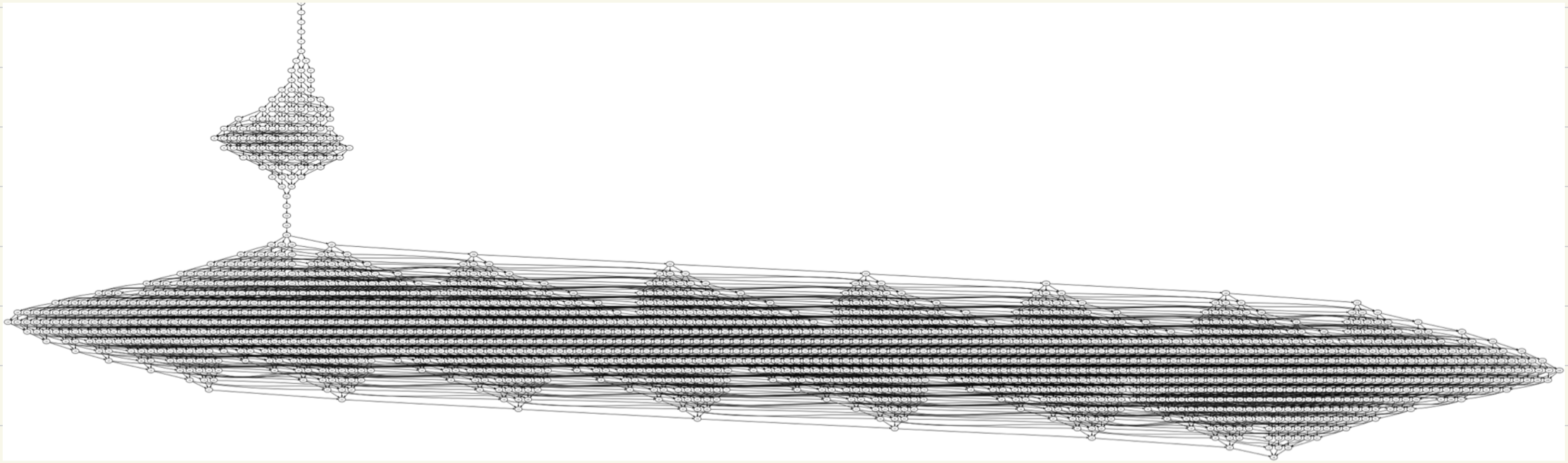
TLB shootdown

[Thread modularity on many levels, Hoenicke, Majumdar, Podelski] POPL'17

1 writer 2 readers



POR w Reach



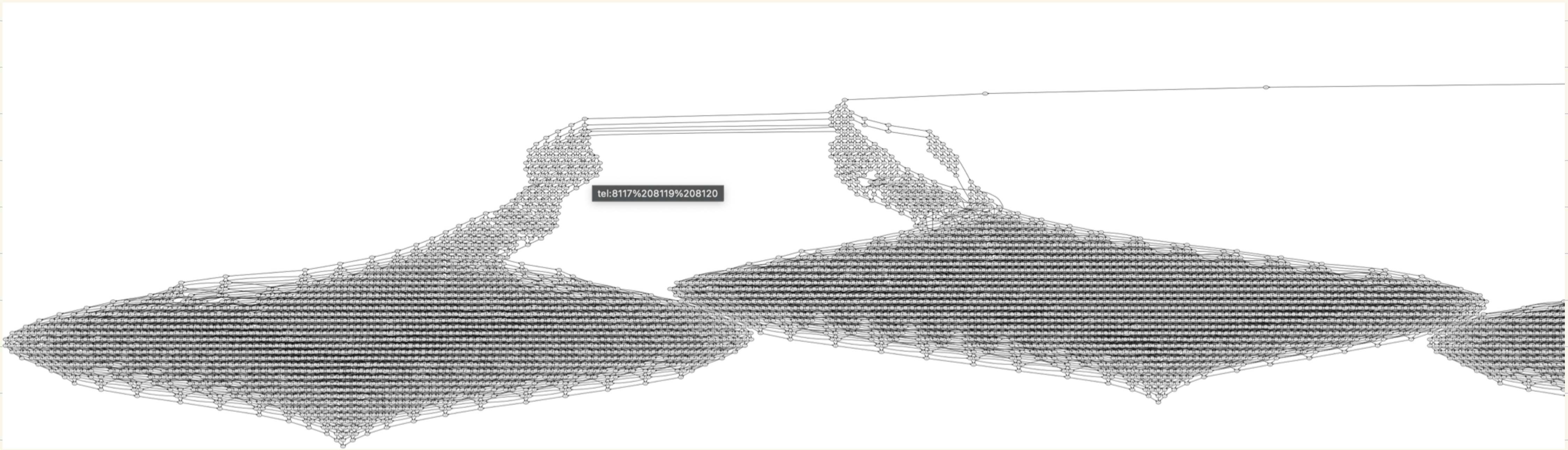
Potential applications

- looking at transition systems
- proving correctness
- verification of timed or probabilistic systems

TLB shootdown

2 writers 1 reader

← POR vs Reach



Concurrent systems

Concurrent programs

write(x, i) await(x, I) acq(ℓ) rel(ℓ)

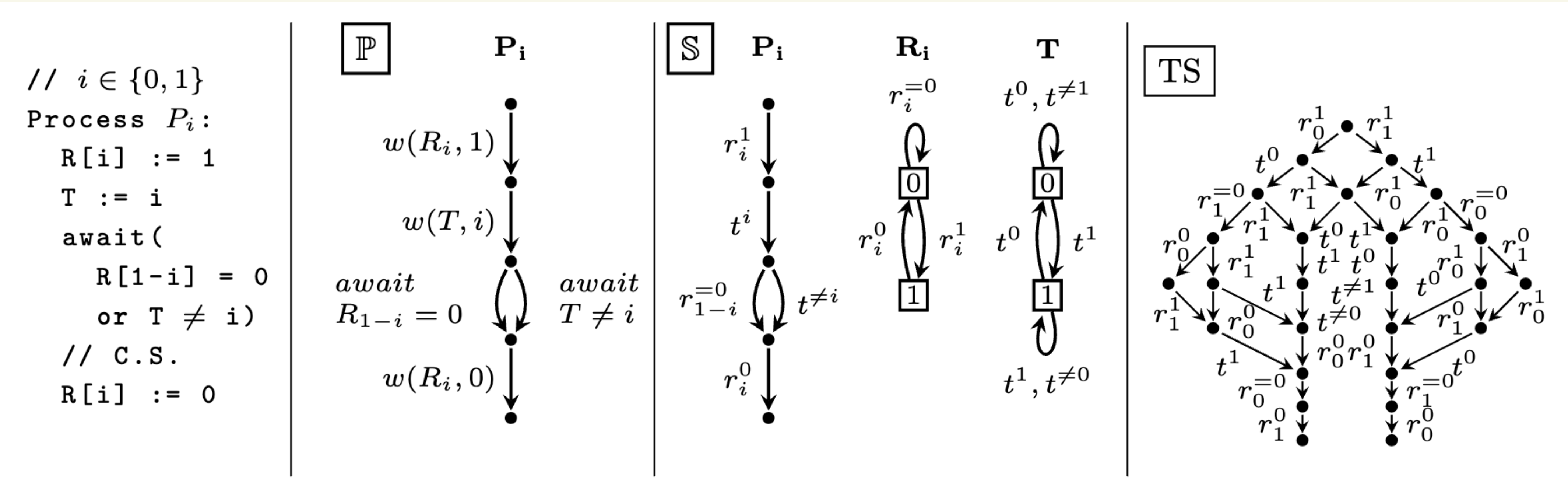
every process is acyclic

Concurrent systems

$\{a, b, \ldots\}$ abstract actions

A process is an acyclic deterministic transition system

Synchronization on common actions



Partial-order reduction

$\approx$ - equivalence relation on sequences of actions

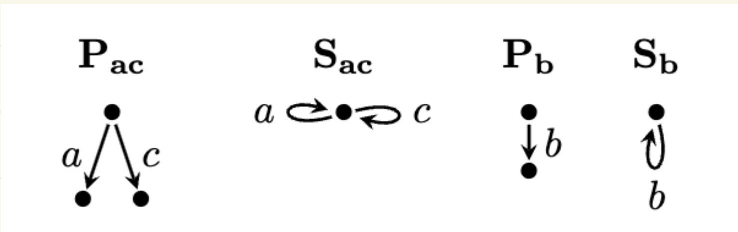
- $r_1(x,15)\ w_2(y,13)\ r_3(x,15) \approx r_3(x,15)\ r_1(x,15)\ w_2(y,13)$

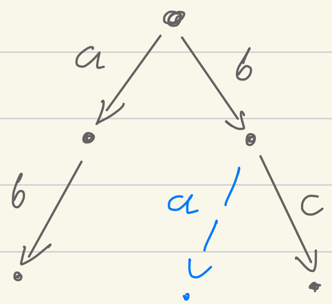
- $a\,b \approx b\,a$ if $dom(a) \cap dom(b) = \emptyset$

TS_r is a <u>reduced transition system</u> for TS if

- every full run of TS_r is a full run of TS
- $\forall$ full run u of TS $\exists$ full run v of TS_r $u \approx v$

<u>Goal</u>: construct a reduced transition system





Partial-order reduction

$\approx$ - equivalence relation on sequences of actions

- $r_1(x,15)\ w_2(y,13)\ r_3(x,15) \approx r_3(x,15)\ r_1(x,15)\ w_2(y,13)$
- $a\,b \approx b\,a$ if $dom(a) \cap dom(b) = \emptyset$

TS_r is a <u>reduced transition system</u> for TS if
- every full run of TS_r is a full run of TS
- $\forall$ full run u of TS $\exists$ full run v of TS_r $u \approx v$

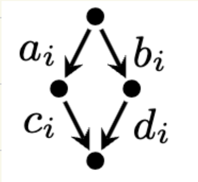
<u>Goal</u>: construct a reduced transition system

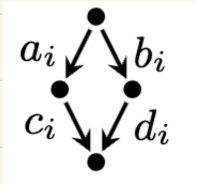
Trace optimality: TS_r has the least number of full paths.

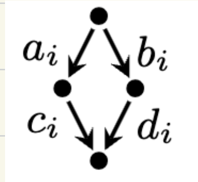
State optimality: TS_r has the least number of states.

Partial-order reduction: example

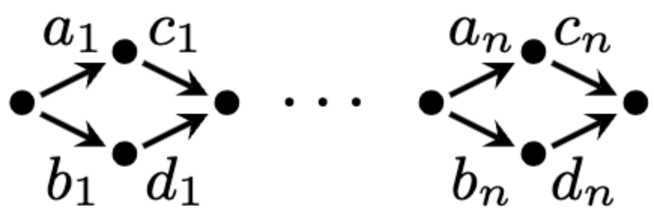
P_1 P_2 ... P_n



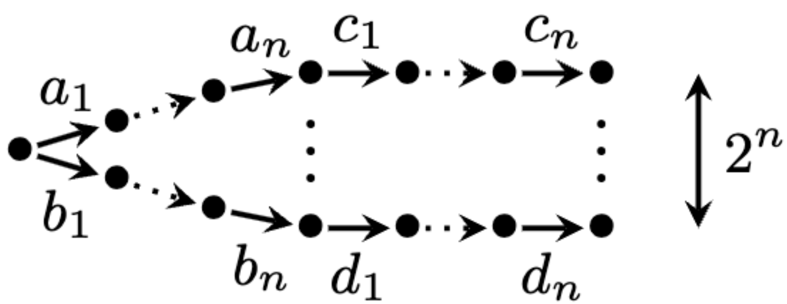


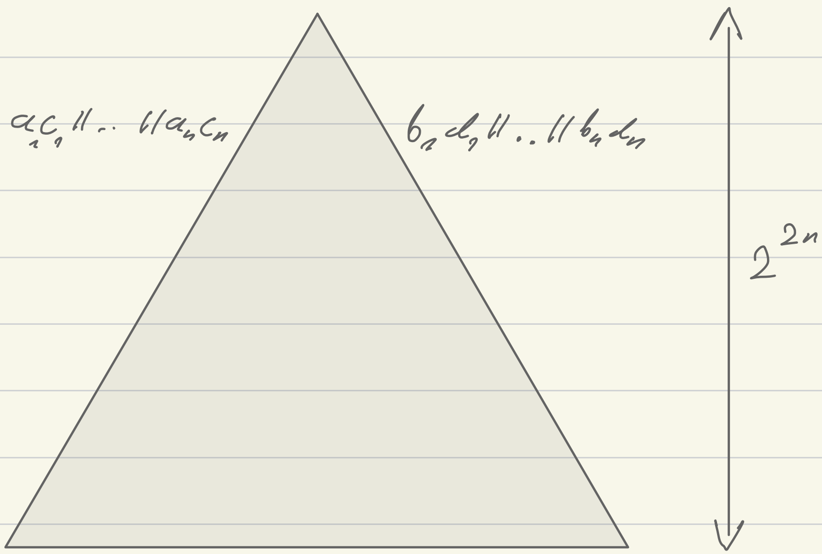


Small transition system



Big transition system





Plan

1. POR is NP-hard
2. An idealized POR algorithm with IFS oracle
3. A heuristic for IFS oracle + implementation

POR is NP-hard

$\min TS(\mathcal{P})$: minimal size of a reduced TS for $\mathcal{P}$

An <u>excellent POR algorithm</u> : constructs TS for $\mathcal{P}$
of size $\leq q(\min TS(\mathcal{P}))$
in time $r(|\mathcal{P}| + \min TS(\mathcal{P}))$

<u>Thm</u>: If $P \neq NP$ then there is no excellent POR algorithm even for programs using only write and await operations.

POR is NP-hard

$\min TS(\mathcal{P})$: minimal size of a reduced TS for $\mathcal{P}$

An <u>excellent POR algorithm</u> : constructs TS for $\mathcal{P}$
of size $\leq q(\min TS(\mathcal{P}))$
in time $r(|\mathcal{P}| + \min TS(\mathcal{P}))$

<u>Thm</u>: If $P \neq NP$ then there is no excellent POR algorithm even for programs using only write and await operations.

Proof: Use 3-SAT. For φ construct S_φ s.t.

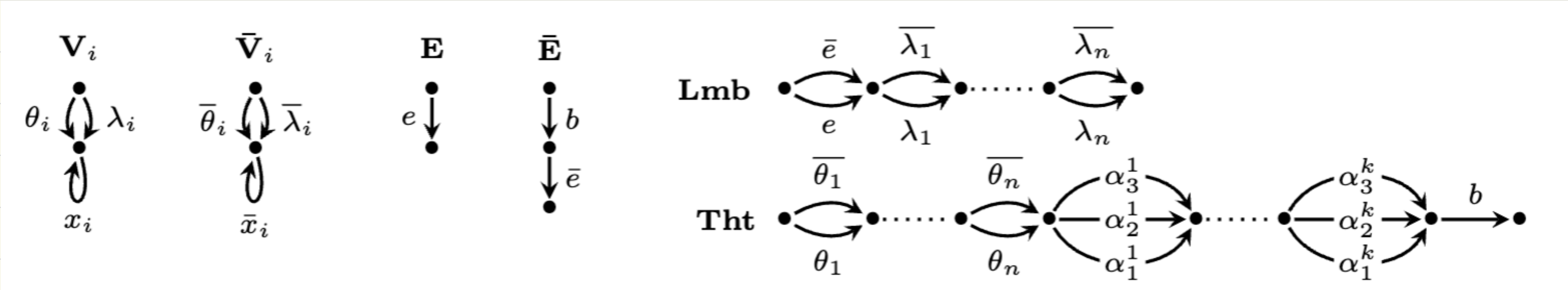
- φ not SAT $\Rightarrow$ $\min TS(S_\varphi) < 6|\varphi|$
- φ SAT $\Rightarrow$ $\min TS(S_\varphi) > \#$ of sat valuations of φ

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

Run hypothetical Alg on S_ψ for $r(6|\psi|)$ time.
If φ not SAT then Alg stops producing a TS for S_ψ
If φ SAT then Alg cannot stop as the smallest TS has $2^m > r(6|\varphi|)$ states.

$\square$

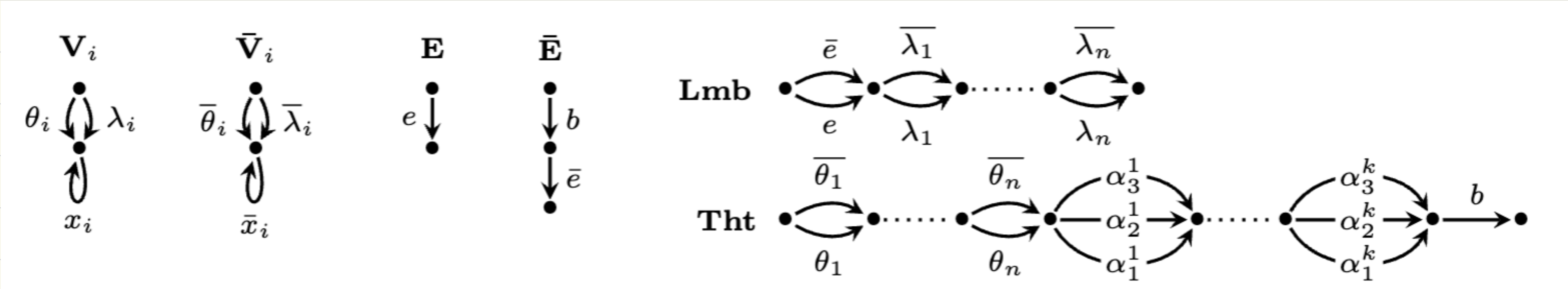
How to construct S_φ



L1: If φ not SAT then all runs of S_φ start with e.

Proof: Otherwise $b\,\bar{e}$ should appear before.
For this we need a sat valuation of φ $\square$

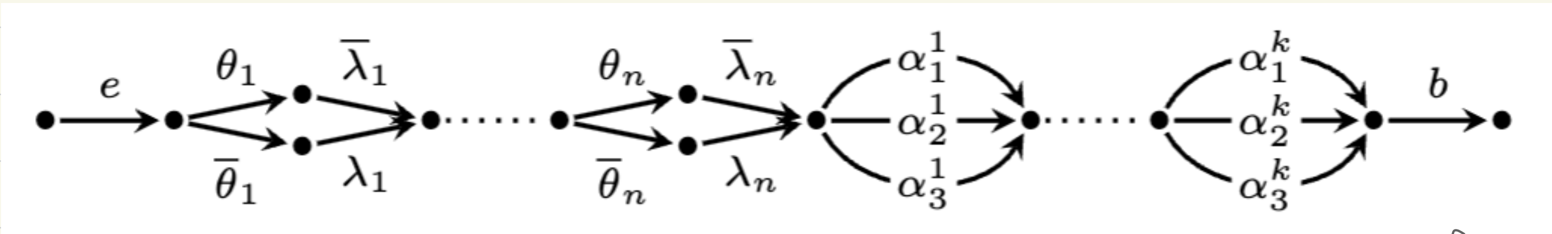
How to construct S_φ



L1: If φ not SAT then all runs of S_φ start with e.

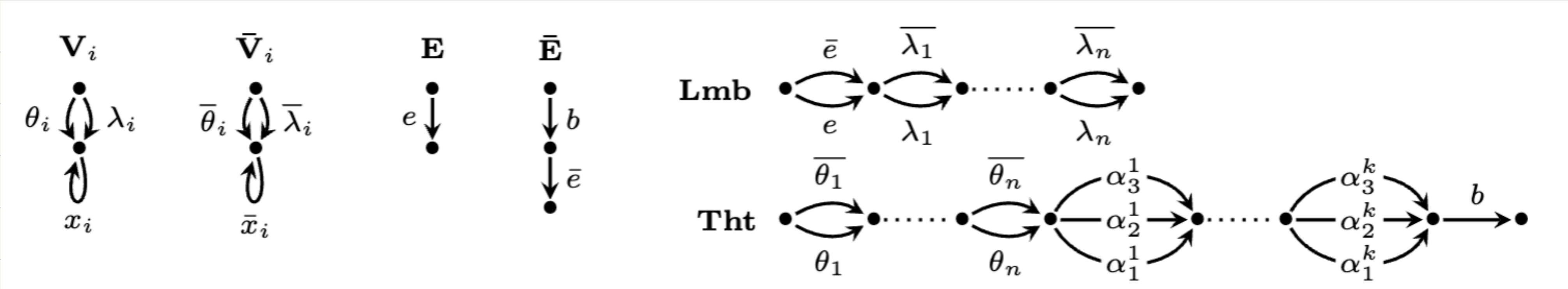
Proof: Otherwise $b\ \bar{e}$ should appear before.
For this we need a sat valuation of φ $\square$

L2 If φ not SAT then the following is a good TS for S_φ



Every run starting from e is equivalent to a run of this form

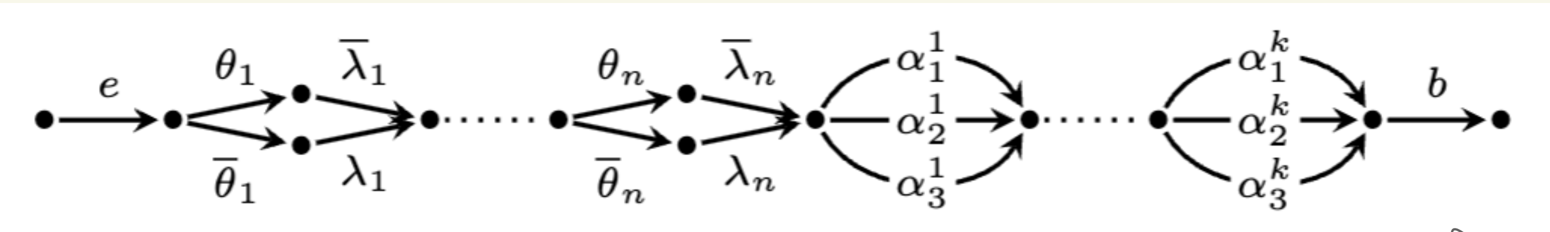
How to construct S_φ



L1: If φ not SAT then all runs of S_φ start with e.

Proof: Otherwise $b\ \bar e$ should appear before.
For this we need a sat valuation of φ □

L2 If φ not SAT then the following is a good TS for S_φ



Every run starting from e is equivalent to a run of this form

L3 If φ SAT then there is a run starting from $\bar e$ for every valuation sat φ.

$$\theta_1\bar\theta_2\dots\theta_n\alpha^1_{i_1}\dots\alpha^k_{i_k}\,b\bar e\bar\lambda_1\lambda_2\dots\bar\lambda_n$$

Plan

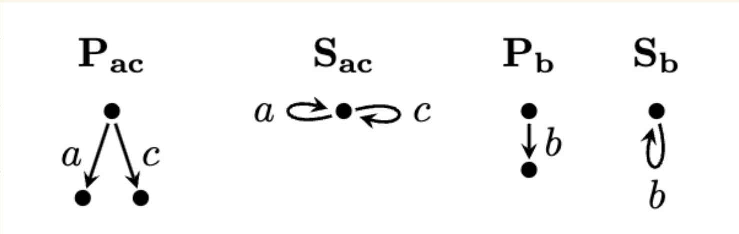
1. POR is NP-hard
2. An idealized POR algorithm with IFS oracle
3. A heuristic for IFS oracle + implementation

First sets

Goal : construct a reduced transition system

$$first(u) = \{ b : \exists v \;\; bv \approx u \}$$

$$First(s) = \{ first(u) : u \text{ a maximal run from } s \}$$

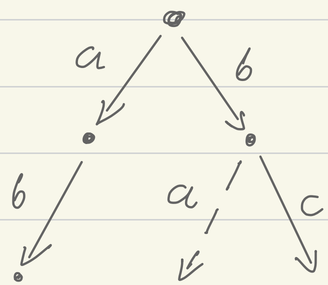


$first(ab) = \{a, b\}$ $\quad First(s^0) = \{\{a,b\}\{c,b\}\}$

Def B is a source set in s if $B \cap F \neq \emptyset$ for every $F \in First(s)$.

Prop It is enough to find B_s for every s and explore only B_s from s.

Rem Source sets are not enough even for trace optimality.



$\{c\}$ is not a source set

Idealized algorithm, IFS oracle

IFS(s, B) includes first set : $\exists$ $s \xrightarrow{u}$ full run first(u) $\subseteq B$

procedure *TreeExplore*(n):

  $Sl := sleep(n)$ // invariant: $Sl = sleep(n) \cup$ {labels of transitions outgoing from n}

  while $enabled(n) - Sl \neq \emptyset$

    choose smallest $e \in (enabled(n) - Sl)$ w.r.t. linear ordering on actions

    let s' **such that** $s(n) \xrightarrow{e} s'$ in $\mathrm{TS}(\mathbb{S})$

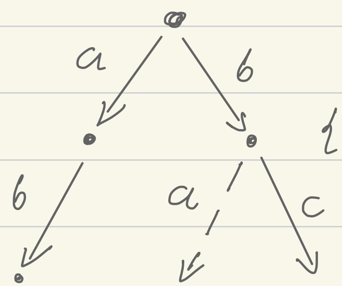
    if $IFS(s', enabled(s') - (Sl - De))$

      create node n' with $s(n') = s'$ **and** $sleep(n') = Sl - De$

      add edge $n \xrightarrow{e} n'$

      TreeExplore(n')

    $Sl := Sl \cup \{e\}$

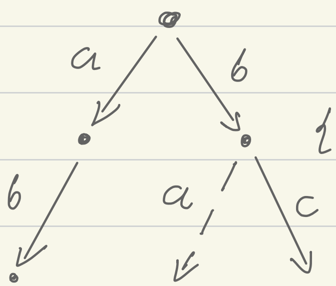


{a} as aIb

Idealized algorithm, IFS oracle

$IFS(s, B)$ includes first set : $\exists\ s \xrightarrow{u}$ full run first$(u) \subseteq B$

procedure *TreeExplore*(n):
  $Sl := sleep(n)$ // invariant: $Sl = sleep(n) \cup \{$labels of transitions outgoing from $n\}$
  while $enabled(n) - Sl \neq \emptyset$
    choose smallest $e \in (enabled(n) - Sl)$ w.r.t. linear ordering on actions
    let s' **such that** $s(n) \xrightarrow{e} s'$ in $\mathrm{TS}(\mathbb{S})$
    if $IFS(s', enabled(s') - (Sl - De))$
      create node n' with $s(n') = s'$ **and** $sleep(n') = Sl - De$
      add edge $n \xrightarrow{e} n'$
      TreeExplore(n')
    $Sl := Sl \cup \{e\}$



$\{a\}$ as aIb

Fact: This algorithm constructs a trace optimal tree

Fact: $IFS(s, B)$ test is NP-hard.

Plan

1. POR is NP-hard
2. An idealized POR algorithm with IFS oracle
3. A heuristic for IFS oracle + implementation

A heuristic for $IFS(s, B)$

Concurrent programs

$write(x, i)$ $\quad read(x, I)$ $\quad acq(\ell)$ $\quad rel(\ell)$

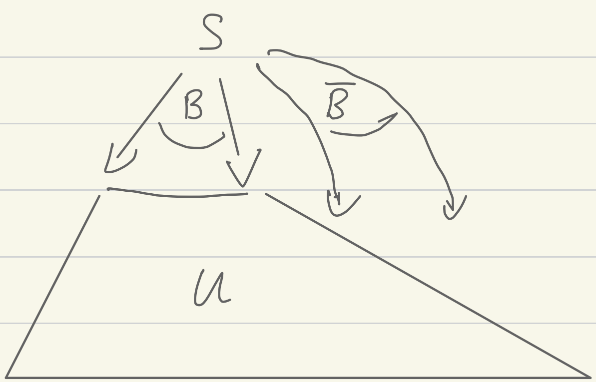
every process is acyclic

Independence

$r_p(x, I) \; I \; r_q(x, J)$ if $I \cap J \neq \emptyset$

$w_p(x, i) \; I \; w_q(x, i)$

$IFS(s, B)$



$first(u) \subseteq B$

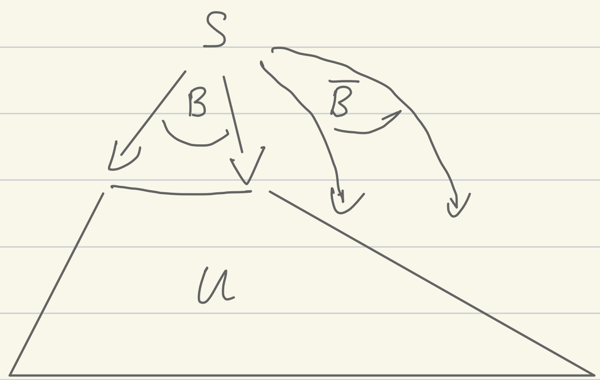
$\forall a \in \overline{B} \quad \exists\, b_a \in u \qquad a \, D \, b_a$

$r_p(x, 5) \; D \; w_q(x, 5)$

$w_p(x, 1) \; D \; r_q(x, 0)$ ⇐ initial read

A heuristic for IFS(s, B)

IFS(s, B)



$first(u) \subseteq B$

$\forall a \in \bar{B} \quad \exists\, b_a \in u \qquad a\ D\ b_a$

$r_p(x,5)\ D\ w_q(x,\bar{5})$

$w_p(x,1)\ D\ r_q(x,0)$ ← initial read

We want to compute:

$$(AIR, APW) = \{IR(u), PW(u)) : \ u \text{ a full run from } s,\ first(u) \subseteq B\}$$

AIR: all initial reads

APW: all produced writes

Signatures

$$Sig(u) = (IR, NR, PW)$$

initial reads ↑ needed reads ↑ produced writes

$$Sig(r_p(x,0)\, w_p(x,1)) = (\{r(x,0)\}, \emptyset, \{w(x,1)\})$$

$$Sig(r_q(x,1)\, w_q(x,2)) = (\emptyset, \{r(x,1)\}, \{w(x,2)\})$$

$$Sig(w_p(x,1)\, r_p(x,2)) = (\emptyset, \{r(x,2)\}, \{w(x,1)\})$$

For $u \equiv w_p(x,1)\, r_q(x,1)\, w_q(x,2)\, r_p(x,2)$ $\quad Sig(u) = (\emptyset, \emptyset, \{w(x,1), w(x,2)\})$

There is an operation $Sig(u{\downarrow}p) \oplus Sig(u{\downarrow}q) = Sig(u)$

We will precompute $Sigs_p(s_p, b)$ for every process p and $s_p \xrightarrow{b}$ in p

$$Sigs_p(s_p, b) = \{ Sig(u) : s_p \xrightarrow{b} \xrightarrow{u} \text{ a run} \}$$

Computing AIR, APW

$$Sig(u) = (IR, NR, PW)$$

initial reads ↑ needed reads ↑ produced writes

$$(AIR_{s,B}, APW_{s,B}) = \{ (IR(u), PW(u)) : u \text{ a full run from } s,\ first(u) \subseteq B \}$$

Algorithm: • precompute $Sig_{s_p}(s_p, b)$ for every process p and $s_p \xrightarrow{b}$ in p
• use these to compute $(AIR_{s,B}, APW_{s,B})$

B: p	B: q	r
$w(x,1)$ ↓	$r(x,1)$ ↓	$w(y,4)$ ↓
$r(x,2)$ ↓	$w(x,2)$ ↓	
$w(y,3)$ ↓		

1) $w(x,1) \in APW_{s,B}$
2) $w(x,2) \in APW_{s,B}$
3) $w(y,3) \in APW_{s,B}$

$w(y,3)\ D\ w(y,4)$
so $IFS(s,B)$ holds

Computing AIR, APW

$$Sig(u) = (IR, NR, PW)$$

initial reads — needed reads — produced writes

$$(AIR_{s,B}, APW_{s,B}) = \{ (IR(u), PW(u)) : u \text{ a full run from } s,\ first(u) \subseteq B \}$$

Algorithm: • precompute $Sig_{s_p}(s_p, b)$ for every process p and $s_p \xrightarrow{b}$ in p

• use these to compute $(AIR_{s,B}, APW_{s,B})$

p (B)	q (B)	r
$w(x,1)\downarrow$	$r(x,1)\downarrow$	$w(y,4)\downarrow$
$r(x,2)\downarrow$	$w(x,2)\downarrow$	
$w(y,3)\downarrow$		

1) $w(x,1) \in APW_{s,B}$
2) $w(x,2) \in APW_{s,B}$
3) $w(y,3) \in APW_{s,B}$

$w(y,3)\ D\ w(y,4)$
so $IFS(s,B)$ holds

Rem: This is only an approximation

p'
$r(x,2)\downarrow$
$w(x,1)\downarrow$
$w(y,3)\downarrow$

gives the same result
this time it is wrong

Our approach

1) Do pre-processing: fully explore each process to compute signatures

2) At each state do poly(|P|) work to compute required PIFS(s, B)

3) Do not traverse constructed TS more than DFS does

TS grows exponentially wrt to |P| while we can hope to keep poly(|P|) small

$$|TS|^{3/2} \gg \text{poly}(|P|) \cdot |TS|$$

Experiments

× 1000 states

seconds

	N. persist	N. PIFS	Ratio	T. persist	T. PIFS
Bakery-barrier-asym, 5	3387	90	38	38	8
Bakery-late-ticket, 5	8762	225	39	136	7
Peterson-priority, 6	4447	818	5	58	23
Philosophers-abandon-order, 15		53			3
Philosophers-barrier, 6, 2	4741	476	10	40	11
Szymanski, 5, 2	11242	958	12	114	19
Szymanski, 6, 1	9532	476	20	40	11
Token-ring, 5, 1	1435	746	2	17	17

on my laptop: M1, 16 GB RAM

Conclusions

- POR is computationally difficult, so we need heuristics.
- We do not have a convienient success measure but state optimality.
- Our heuristic is quite satisfactory for variables and locks.

TODO

1) Lower bound for stateless POR.
2) PIFS for channels instead of variables.
3) Symmetry reduction.