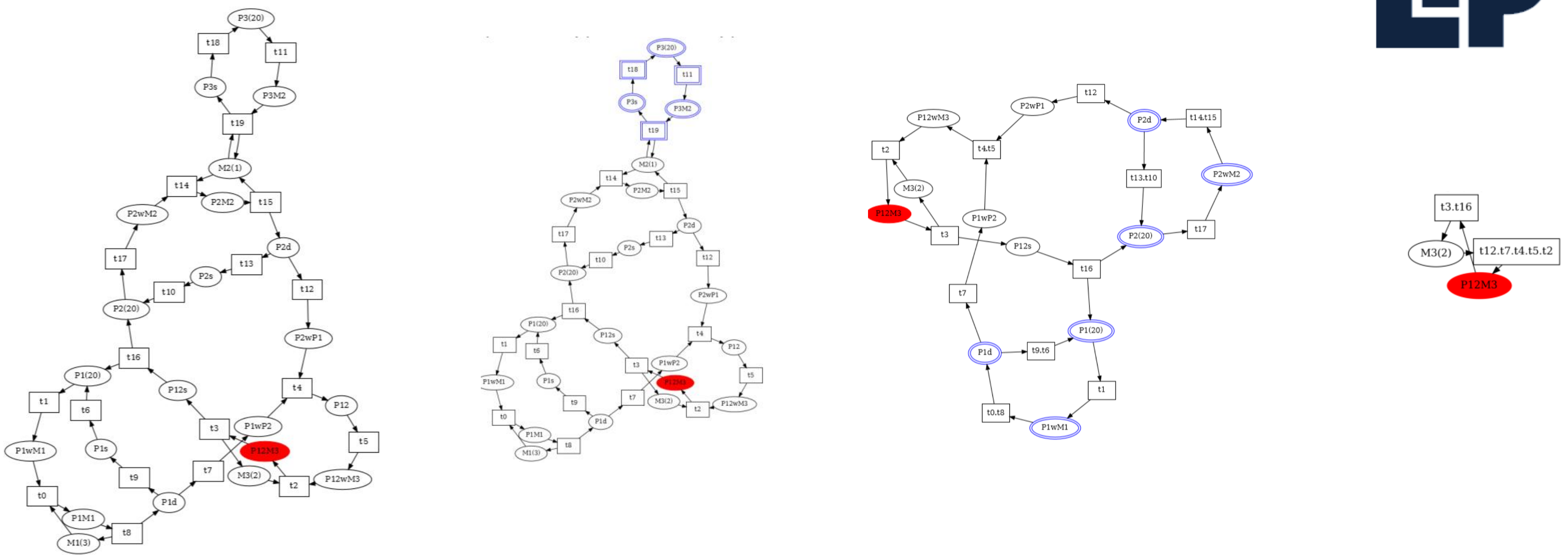


EFFICIENT STRATEGIES TO COMPUTE INVARIANTS, BOUNDS AND STABLE PLACES OF PETRI NETS

Yann Thierry-Mieg

LIP6, Sorbonne Université, CNRS

PNSE'23: Petri Nets and Software Engineering 2023, June 2023, Lisbon

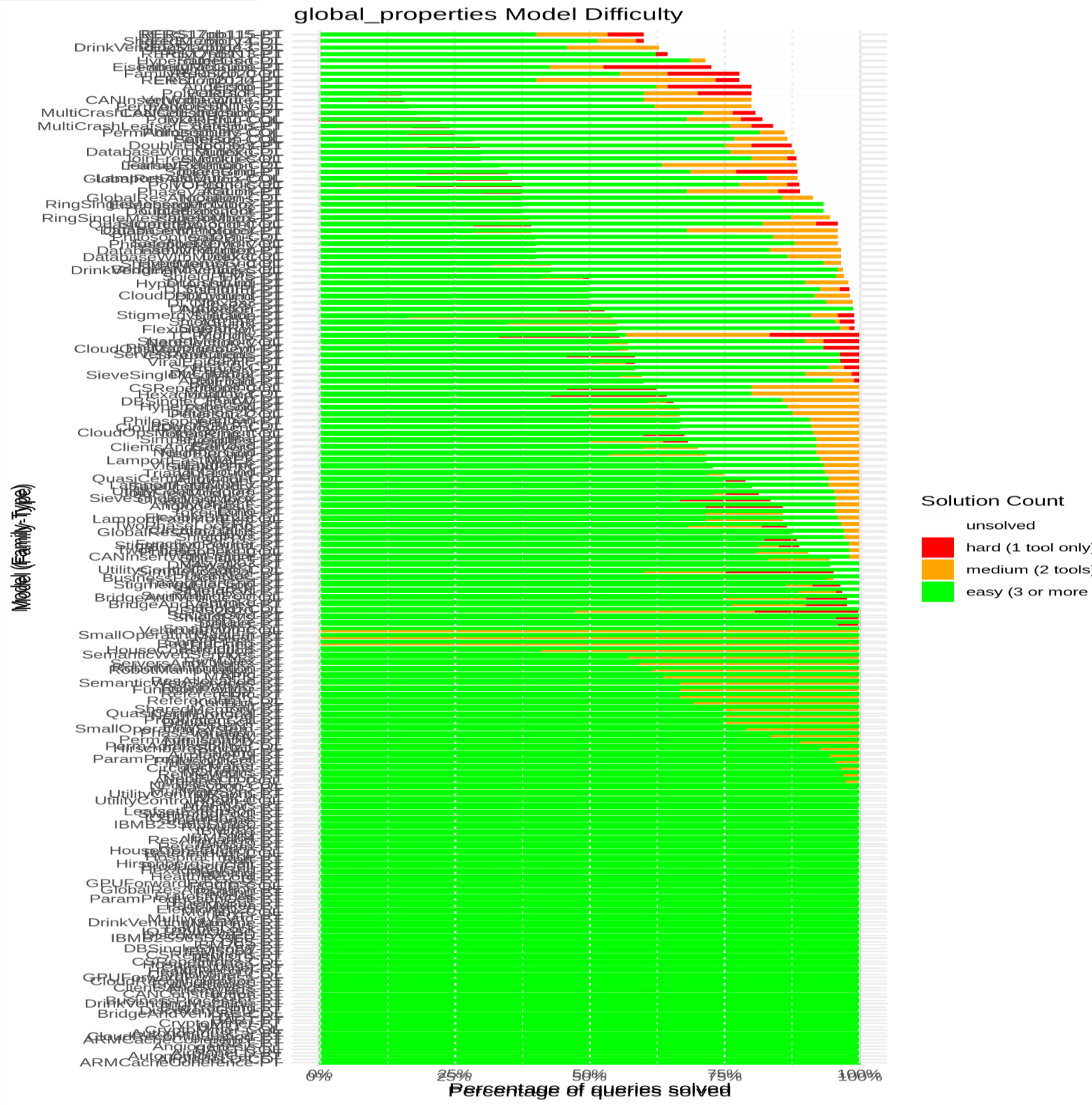
CONTEXT

The Model Checking Contest

- Annual event since 2011, comparing verification tools
- Large benchmark : 1678 instances from 132 families
- Several examinations
  - State Space : metrics on the size of the state space
  - Upper Bounds : compute upper bound of place markings
  - Reachability, CTL, LTL
- Since 2020, « Global Properties » category
  - One Safe : all places are bounded by one
  - Stable Marking : there exists a (stable) place with a constant (stable) marking
  - Reachability of a Deadlock, Quasi-Liveness, Liveness
- In this presentation state of the art strategies for :
  - Computing Invariants (preliminary)
  - Computing Upper Bounds
  - Computing One Safe and Stable Marking global property in the paper

EXHAUSTIVE STATE SPACE EXPLORATION

- Full state space generation possible for ~70% of model instances with advanced strategies
  - 73% for Best “Virtual” tool
  - 69% for Gold medalist Tedd
- Most competitors use more diverse strategies
  - One Safe : BVT 100%, Gold ITS-Tools 99.88%
  - Stable Marking : BVT 96.6%, Gold ITS-Tools 94.3%
  - Upper Bounds : BVT 94.91%, Gold ITS-Tools 93.59%



https://yanntm.github.io/MCC-analysis/hardness.html

PETRI NETS

A model with strong structural/analytic results

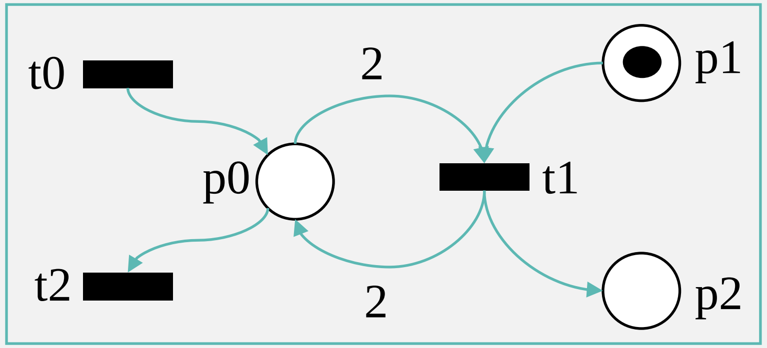
- Places, transitions, arcs, initial marking
- Flow matrices

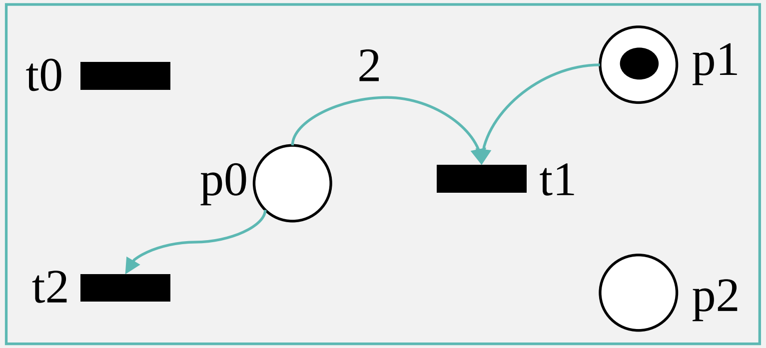
Pre	t0	t1	t2
p0	0	2	1
p1	0	1	0
p2	0	0	0

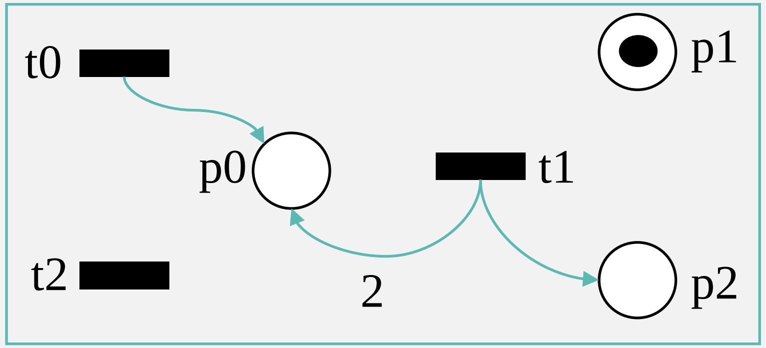
Post	t0	t1	t2
p0	1	2	0
p1	0	0	0
p2	0	1	0

- Transition effects : Post -Pre

Post-Pre	t0	t1	t2
p0	1	0	-1
p1	0	-1	0
p2	0	1	0







PETRI NETS

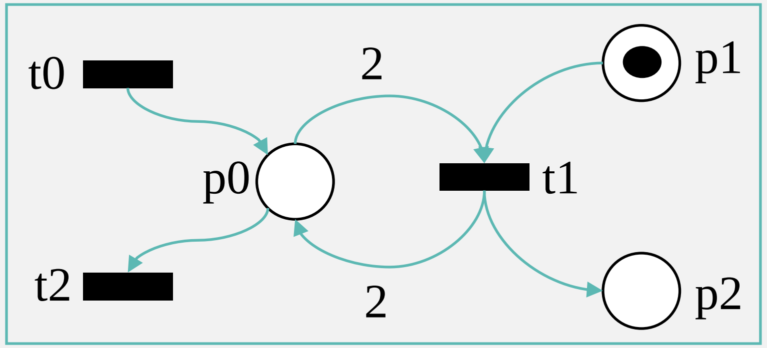
A model with strong structural/analytic results

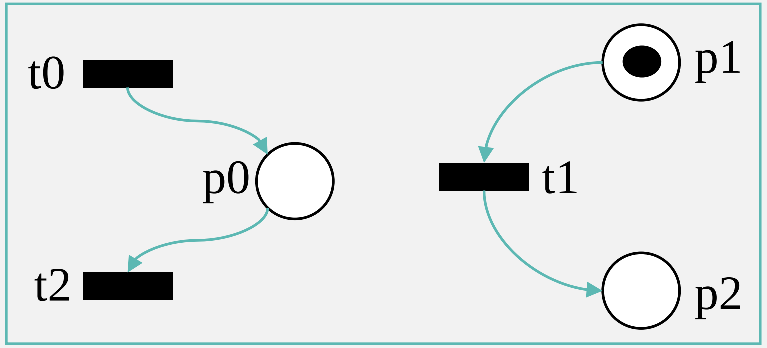
- Places, transitions, arcs, initial marking
- Flow matrices
- Transition effects : Post –Pre

Post-Pre	t0	t1	t2
p0	1	0	-1
p1	0	-1	0
p2	0	1	0

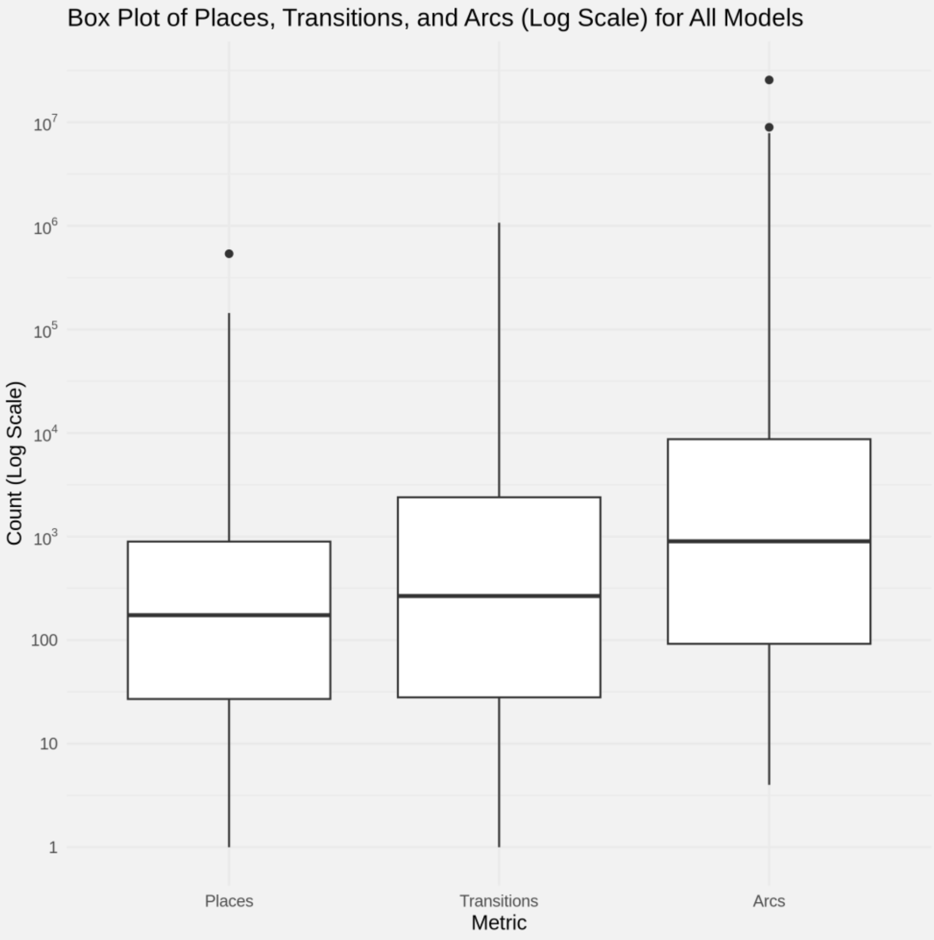
An over approximation that can still yield

- Invariants : p1 + p2 = 1
- The state equation
  - p0 = |t0| - |t2|
  - p1 = 1 - |t1|
  - p2 = |t1|





MCC MODELS CAN BE VERY LARGE



- Models can have up to
  - ~10^5 places
  - ~10^6 transitions
  - ~10^7 arcs
  - But they are mostly **sparse**
- Colored models are small
  - < 100 places, transitions, < 500 arcs
  - But their « unfolding » is huge, larger than most native PT
  - Use the **skeleton** over-approximation when possible

https://yanntm.github.io/MCC-analysis/models/models.html

SPARSE REPRESENTATIONS

- Dense Array N elements
  - Memory O(N)
  - Count of non zero elements, Test Emptiness O(N)
  - Iterate non empty elements O(N)
  - Random access by index O(1)

0	0	0	-1	2	0	0

- Sparse Array with K non zero elements
  - Memory O(2K)
  - Count of non zero elements, Test Emptiness O(1)
  - Iterate non empty elements O(K)
  - Random access by index O(log2(K))

sz	2	
keys	3	4
values	-1	2

Post-Pre	t0	t1	t2
p0	1	0	-1
p1	0	-1	0
p2	0	1	0

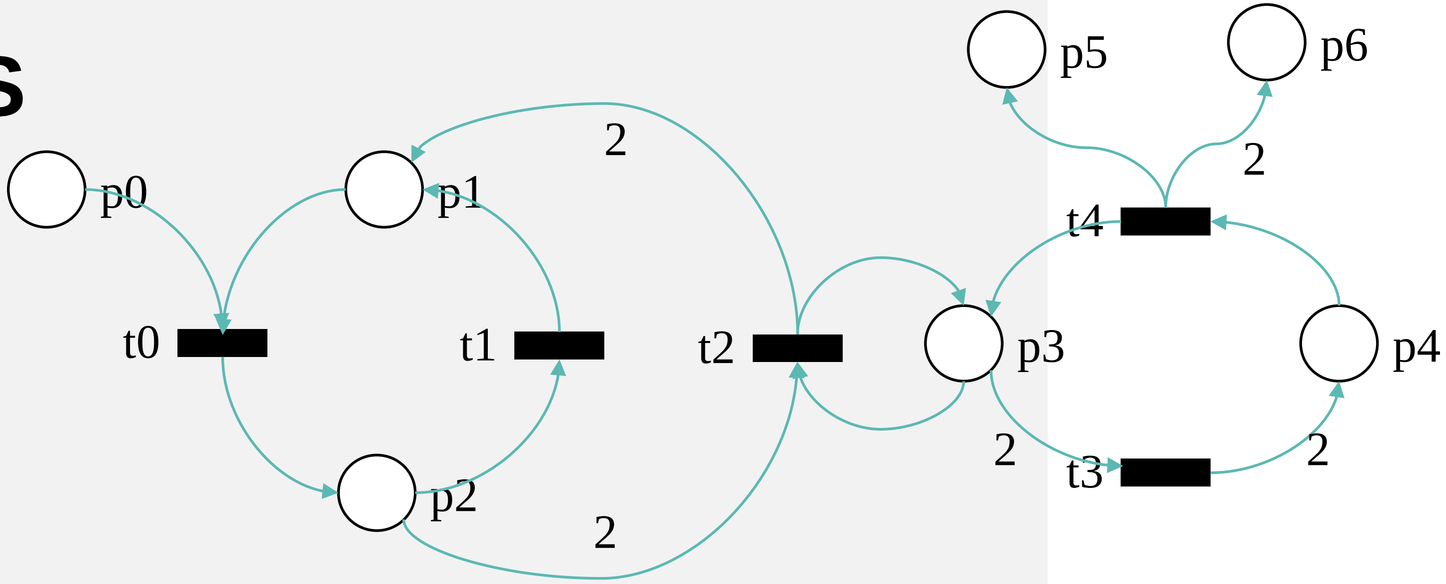
- Sparse Matrix representation
  - Dense Array of Sparse Arrays : column based storage
  - Iterate non zero by column index => very fast

	t0	t1		t2
sz	1	2		1
keys	0	1	2	0
values	1	-1	1	-1

- Clear a column O(1)
- Iterate (non zero) based on row, e.g. clear a row => bad complexity
- But fast and memory efficient transpose available

COMPUTING INVARIANTS

	t0	t1	t2	t3	t4
p0	-1				
p1	-1	1	2		
p2	1	-1	-2		
p3				-2	1
p4				2	-1
p5					1
p6					2

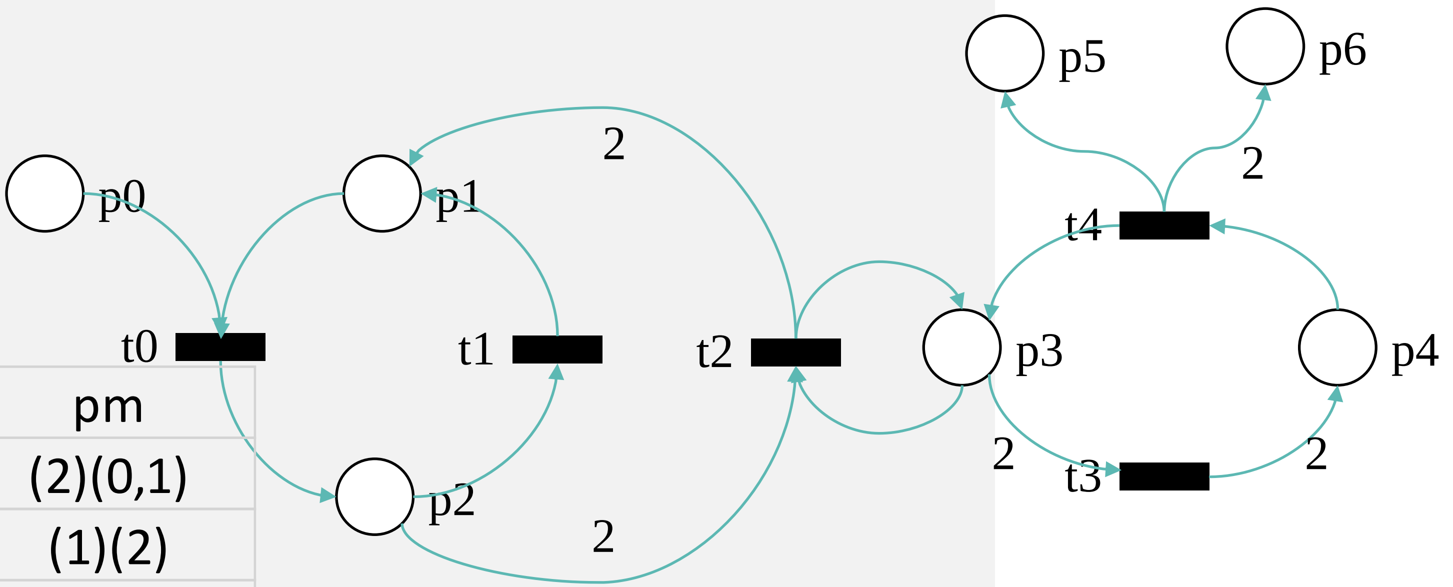


- Step 1 : normalize columns
  - Divide by gcd
  - Eliminate duplicates
  - Transpose

	t0	t1	t2	t3	t4
p0	-1				
p1	-1	1	1		
p2	1	-1	-1		
p3				-1	1
p4				1	-1
p5					1
p6					2

	p0	p1	p2	p3	p4	p5	p6
t0	-1	-1	1				
t1		1	-1				
t3				-1	1		
t4				1	-1	1	2

COMPUTING INVARIANTS

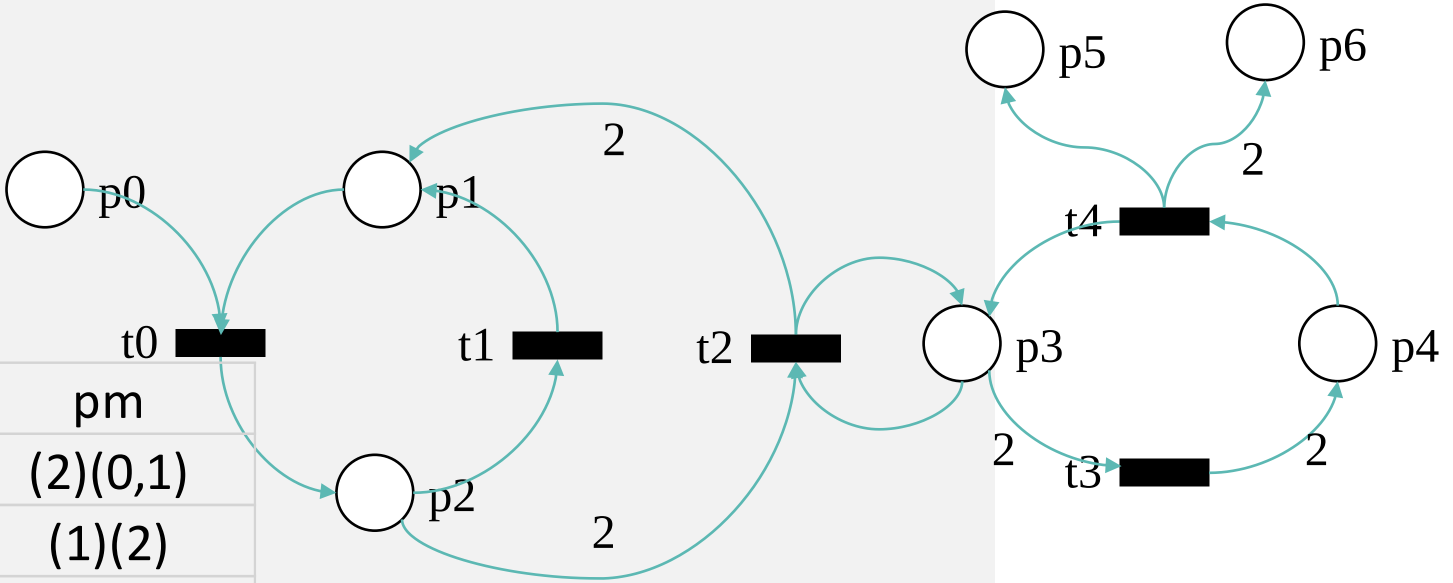


	p0	p1	p2	p3	p4	p5	p6	pm
t0	-1	-1	1					(2)(0,1)
t1		1	-1					(1)(2)
t3				-1	1			(4)(3)
t4				1	-1	1	2	(3,5,6)(4)

	p0	p1	p2	p3	p4	p5	p6
p0	1						
p1		1					
p2			1				
p3				1			
p4					1		
p5						1	
p6							1

- Step 2 :
  - Initialize Identity matrix
  - Initialize pm index per row
    - (list positive indexes)(list negative indexes)

COMPUTING INVARIANTS



	p0	p1	p2	p3	p4	p5	p6	pm
t0	-1	-1	1					(2)(0,1)
t1		1	-1					(1)(2)
t3				-1	1			(4)(3)
t4				1	-1	1	2	(3,5,6)(4)

	p0	p1	p2	p3	p4	p5	p6
p0	1						
p1		1					
p2			1				
p3				1			
p4					1		
p5						1	
p6							1

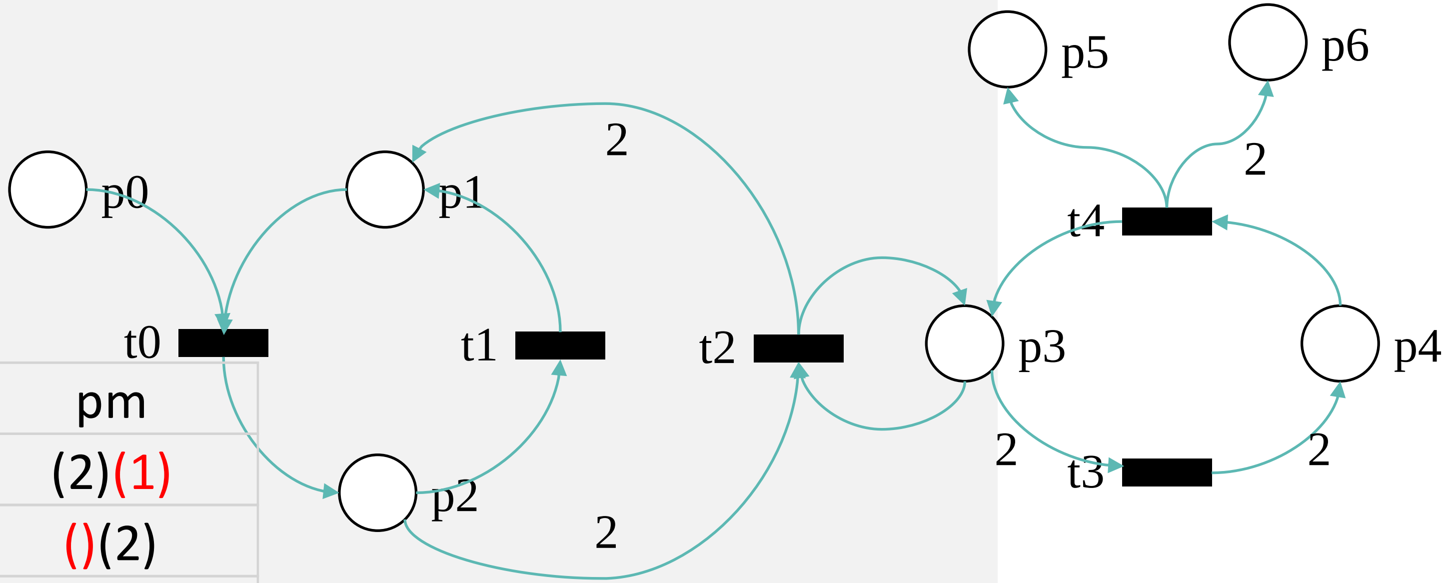
- Step 3 :
  - Find a row « r » with single positive or negative entry « k »
  - For every other column « j » which is non zero on this row
    - Compute coefficient : g=gcd(M[k,r],M[j,r])
    - Add ~g time column k to column j (so M[j,r]=0). NB: this is a sparse operation on two Sparse Array
    - Update pm sparsely.
  - *Clear* column k (sparse)

COMPUTING INVARIANTS

p1=p1+p2

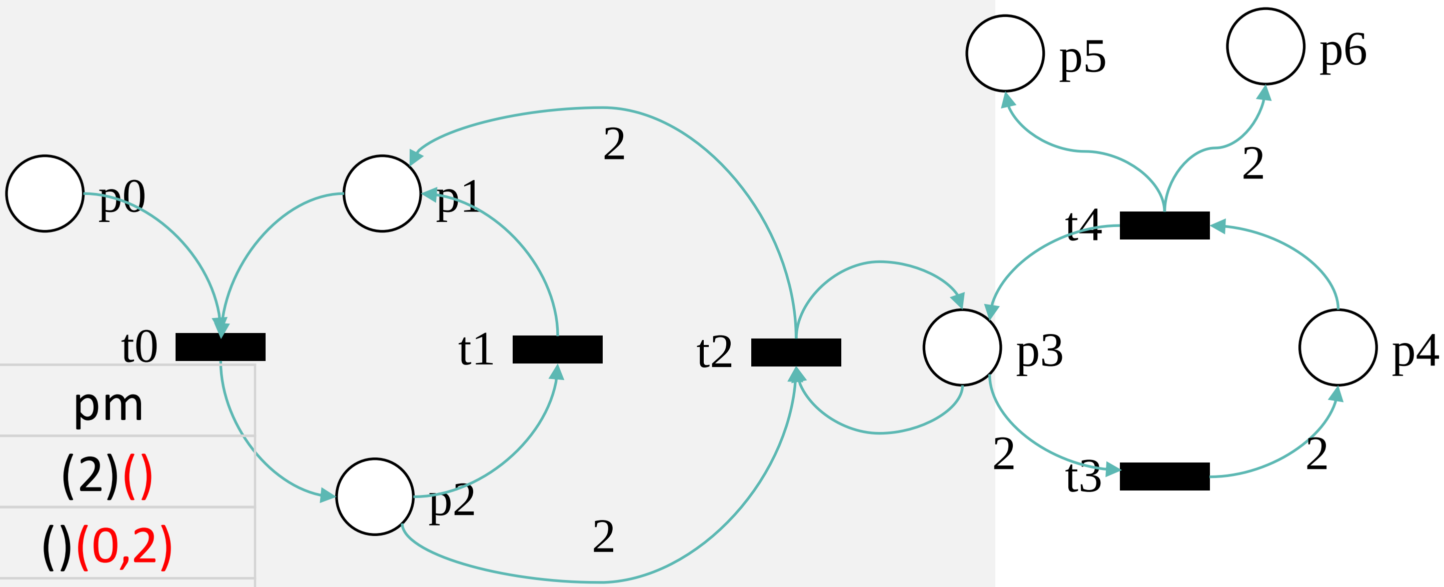
	p0	p1	p2	p3	p4	p5	p6	pm
t0	-1		1					(2)(1)
t1			-1					()(2)
t3				-1	1			(4)(3)
t4				1	-1	1	2	(3,5,6)(4)

	p0	p1	p2	p3	p4	p5	p6
p0	1						
p1		1					
p2		1	1				
p3				1			
p4					1		
p5						1	
p6							1



- Step 3 :
  - Find a row « r » with single positive or negative entry « k »
  - For every other column « j » which is non zero on this row
    - Compute coefficient : g=gcd(M[k,r],M[j,r])
    - Add ~g time column k to column j (so M[j,r]=0). NB: this is a sparse operation on two Sparse Array
    - Update pm sparsely.
  - *Clear* column k (sparse)

COMPUTING INVARIANTS



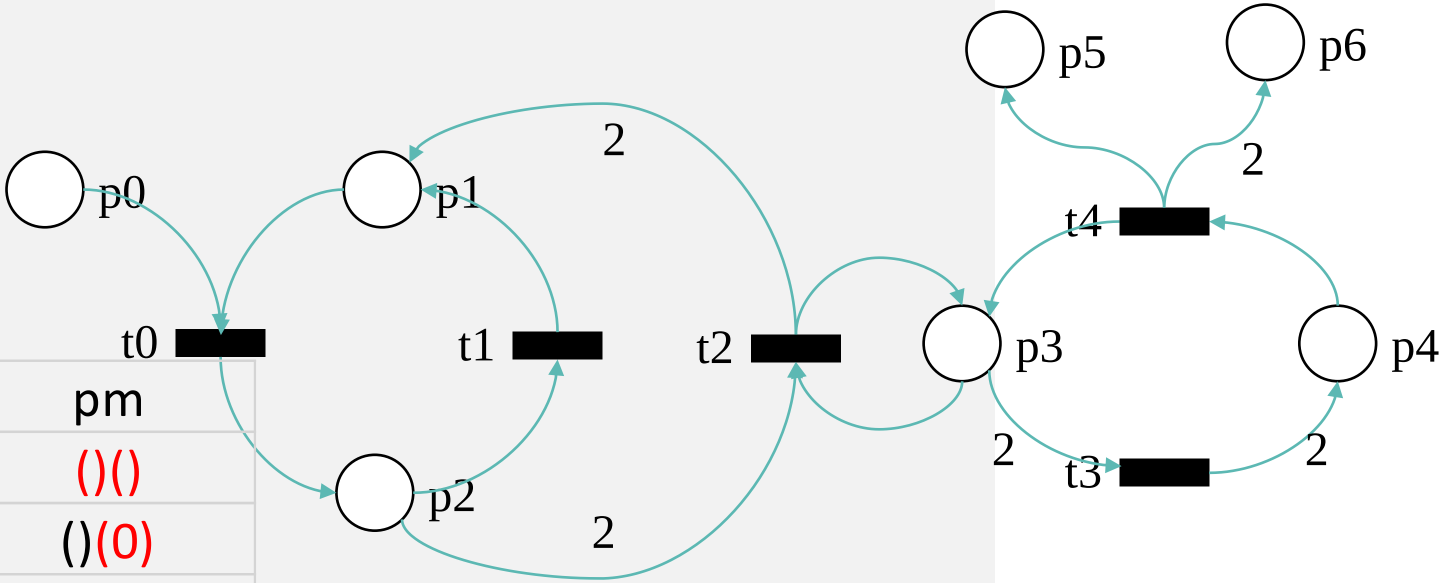
p0=p0+p2

	p0	p1	p2	p3	p4	p5	p6	pm
t0			1					(2)()
t1	-1		-1					()(0,2)
t3				-1	1			(4)(3)
t4				1	-1	1	2	(3,5,6)(4)

	p0	p1	p2	p3	p4	p5	p6
p0	1						
p1		1					
p2	1	1	1				
p3				1			
p4					1		
p5						1	
p6							1

- Step 3 :
  - Find a row « r » with single positive or negative entry « k »
  - For every other column « j » which is non zero on this row
    - Compute coefficient : g=gcd(M[k,r],M[j,r])
    - Add ~g time column k to column j (so M[j,r]=0). NB: this is a sparse operation on two Sparse Array
    - Update pm sparsely.
  - *Clear* column k (sparse)

COMPUTING INVARIANTS

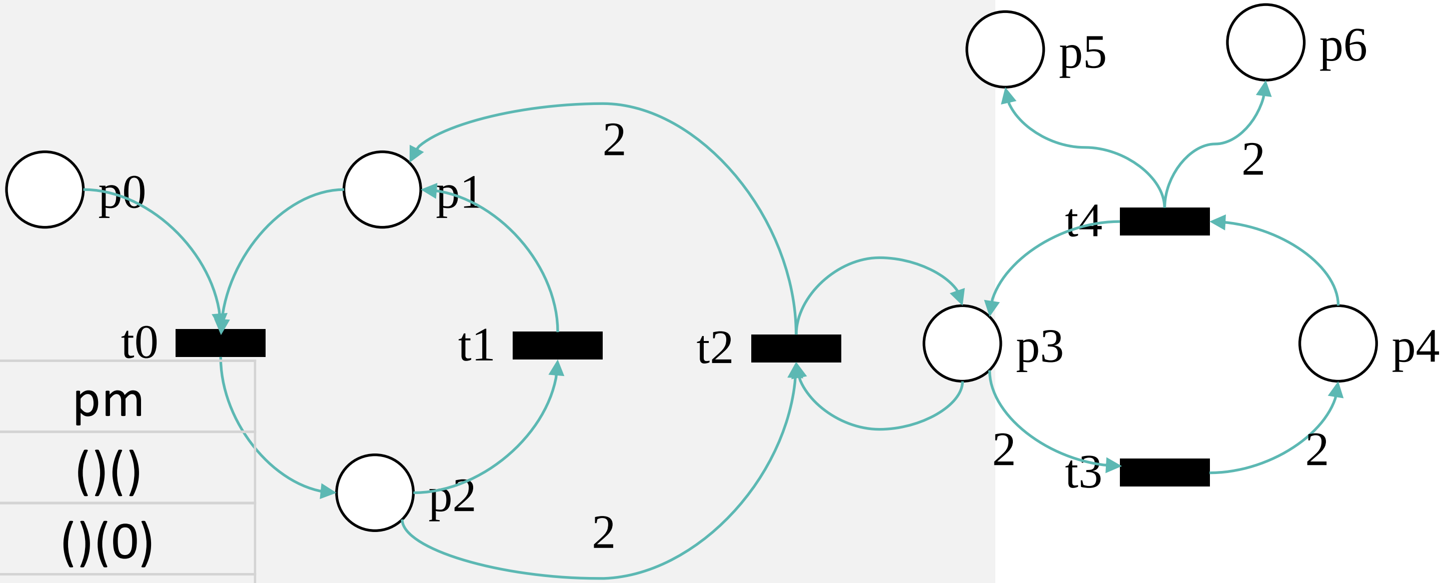


	p0	p1	p2	p3	p4	p5	p6	pm
t0								()()
t1	-1							()(0)
t3				-1	1			(4)(3)
t4				1	-1	1	2	(3,5,6)(4)

	p0	p1	p2	p3	p4	p5	p6
p0	1						
p1		1					
p2	1	1					
p3				1			
p4					1		
p5						1	
p6							1

- Step 3 :
  - Find a row « r » with single positive or negative entry « k »
  - For every other column « j » which is non zero on this row
    - Compute coefficient : g=gcd(M[k,r],M[j,r])
    - Add ~g time column k to column j (so M[j,r]=0). NB: this is a sparse operation on two Sparse Array
    - Update pm sparsely.
  - *Clear* column k (sparse)

COMPUTING INVARIANTS

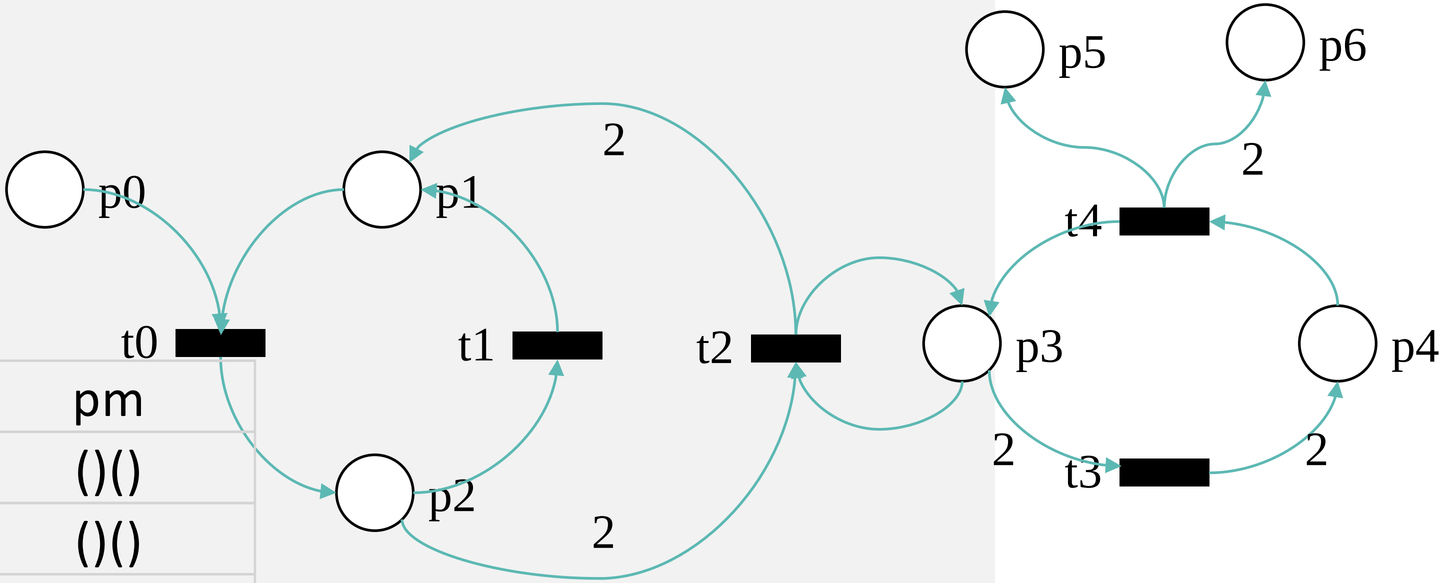


	p0	p1	p2	p3	p4	p5	p6	pm
t0								()()
t1	-1							()(0)
t3				-1	1			(4)(3)
t4				1	-1	1	2	(3,5,6)(4)

	p0	p1	p2	p3	p4	p5	p6
p0	1						
p1		1					
p2	1	1					
p3				1			
p4					1		
p5						1	
p6							1

- Step 3 :
  - Find a row « r » with single positive or negative entry « k »
  - For every other column « j » which is non zero on this row
    - Compute coefficient : g=gcd(M[k,r],M[j,r])
    - Add ~g time column k to column j (so M[j,r]=0). NB: this is a sparse operation on two Sparse Array
    - Update pm sparsely.
  - *Clear* column k (sparse)

COMPUTING INVARIANTS



	p0	p1	p2	p3	p4	p5	p6	pm
t0								()()
t1								()()
t3				-1	1			(4)(3)
t4				1	-1	1	2	(3,5,6)(4)

	p0	p1	p2	p3	p4	p5	p6
p0							
p1		1					
p2		1					
p3				1			
p4					1		
p5						1	
p6							1

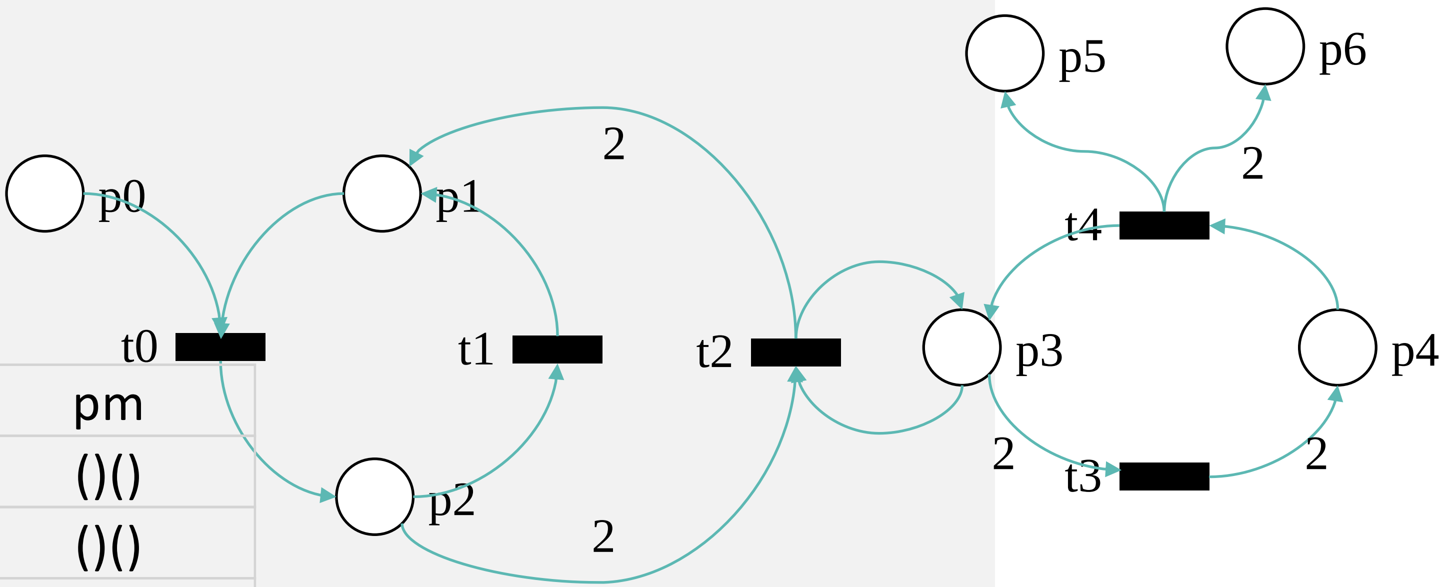
- Step 3 :
  - Find a row « r » with single positive or negative entry « k »
  - For every other column « j » which is non zero on this row
    - Compute coefficient : g=gcd(M[k,r],M[j,r])
    - Add ~g time column k to column j (so M[j,r]=0). NB: this is a sparse operation on two Sparse Array
    - Update pm sparsely.
  - *Clear* column k (sparse)

COMPUTING INVARIANTS

p4=p4+p3

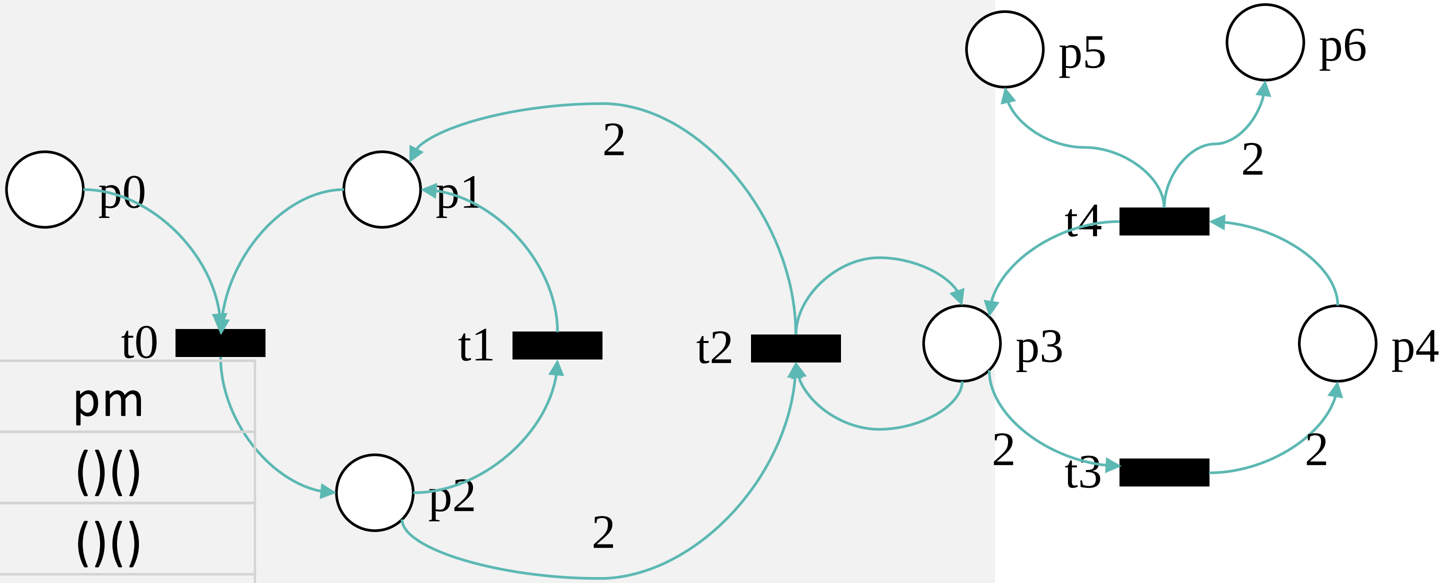
	p0	p1	p2	p3	p4	p5	p6	pm
t0								()()
t1								()()
t3				$\boxed{-1}$	1			(4)(3)
t4				1	-1	1	2	(3,5,6)(4)

	p0	p1	p2	p3	p4	p5	p6
p0							
p1		1					
p2		1					
p3				1			
p4					1		
p5						1	
p6							1



- Step 3 :
  - Find a row « r » with single positive or negative entry « k »
  - For every other column « j » which is non zero on this row
    - Compute coefficient : g=gcd(M[k,r],M[j,r])
    - Add ~g time column k to column j (so M[j,r]=0). NB: this is a sparse operation on two Sparse Array
    - Update pm sparsely.
  - *Clear* column k (sparse)

COMPUTING INVARIANTS



p4=p4+p3

	p0	p1	p2	p3	p4	p5	p6	pm
t0								()()
t1								()()
t3								()()
t4						1	2	(5,6)()

	p0	p1	p2	p3	p4	p5	p6
p0							
p1		1					
p2		1					
p3					1		
p4					1		
p5						1	
p6							1

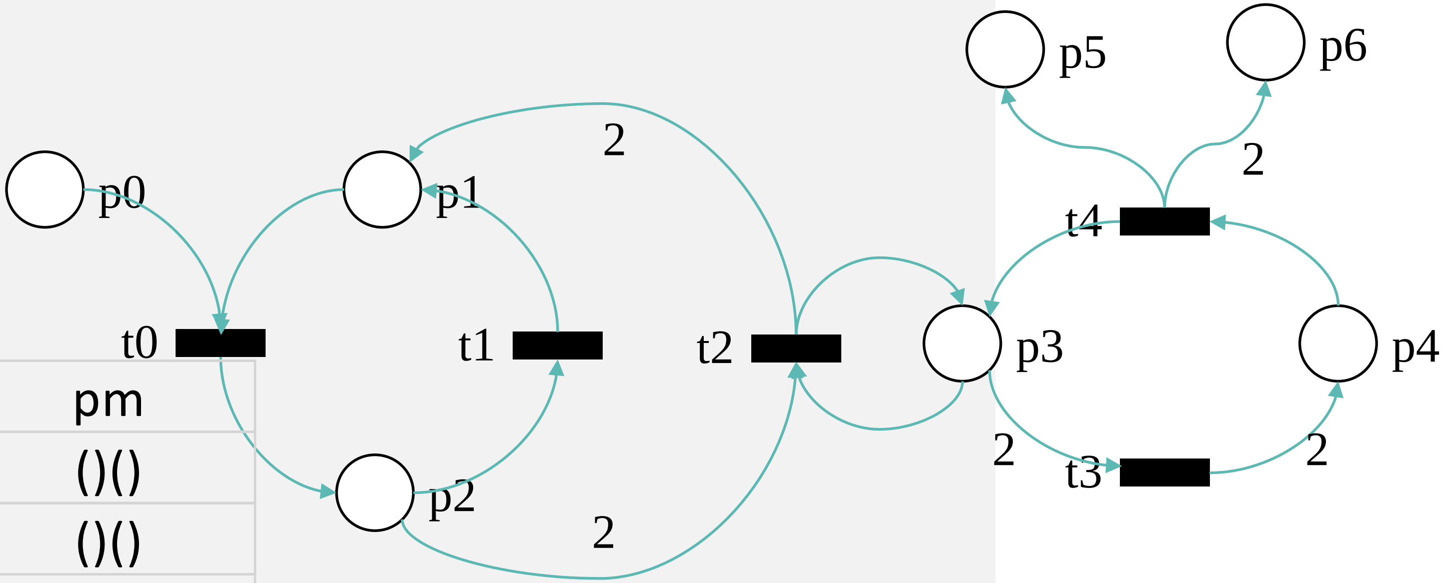
- Step 3 :
  - Find a row « r » with single positive or negative entry « k »
  - For every other column « j » which is non zero on this row
    - Compute coefficient : g=gcd(M[k,r],M[j,r])
    - Add ~g time column k to column j (so M[j,r]=0). NB: this is a sparse operation on two Sparse Array
    - Update pm sparsely.
  - *Clear* column k (sparse)

COMPUTING INVARIANTS

$p6=p6-2*p5$

	p0	p1	p2	p3	p4	p5	p6	pm
t0								()()
t1								()()
t3								()()
t4						1	2	(5,6)()

	p0	p1	p2	p3	p4	p5	p6
p0							
p1		1					
p2		1					
p3					1		
p4					1		
p5						1	
p6							1



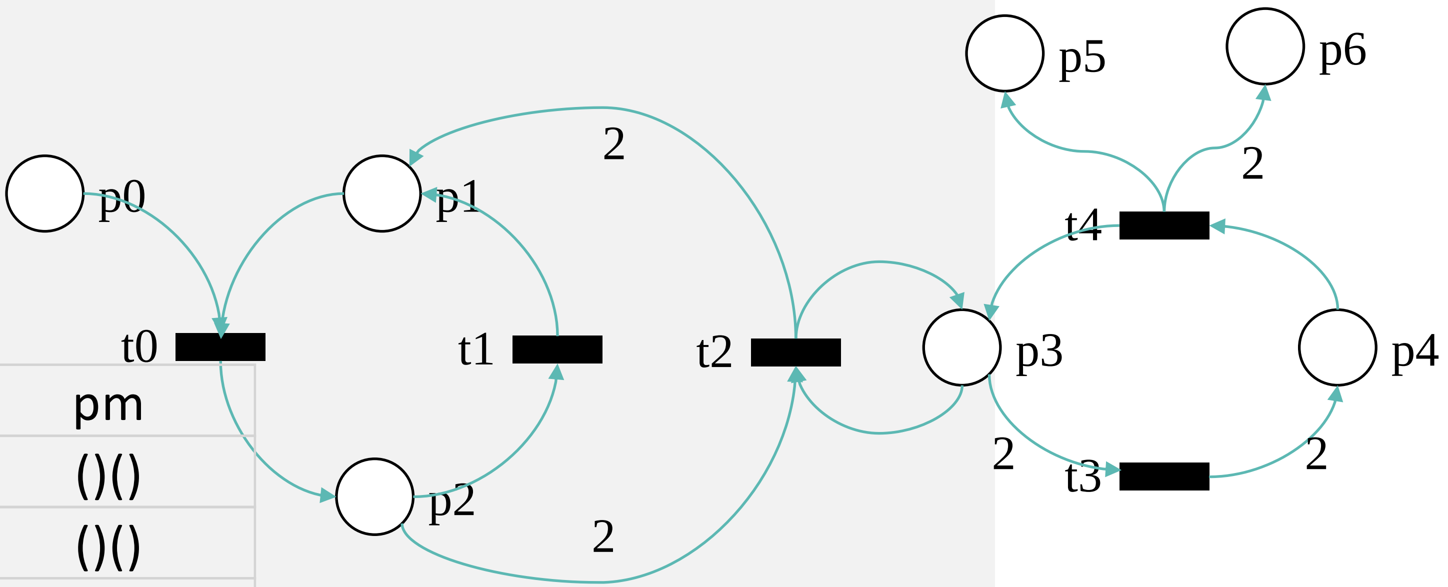
- Step 4 :
  - If no row « r » with single positive or negative entry « k », choose an arbitrary non zero entry
  - For every other column « j » which is non zero on this row
    - Compute coefficient : g=gcd(M[k,r],M[j,r])
    - Add ~(-g or g) times column k to column j to empty cell M[j,r]
    - Update pm (sparsely)
  - Clear column k (sparse)

COMPUTING INVARIANTS

p6=p6-2*p5

	p0	p1	p2	p3	p4	p5	p6	pm
t0								()()
t1								()()
t3								()()
t4								()()

	p0	p1	p2	p3	p4	p5	p6
p0							
p1		1					
p2		1					
p3					1		
p4					1		
p5							-2
p6							1

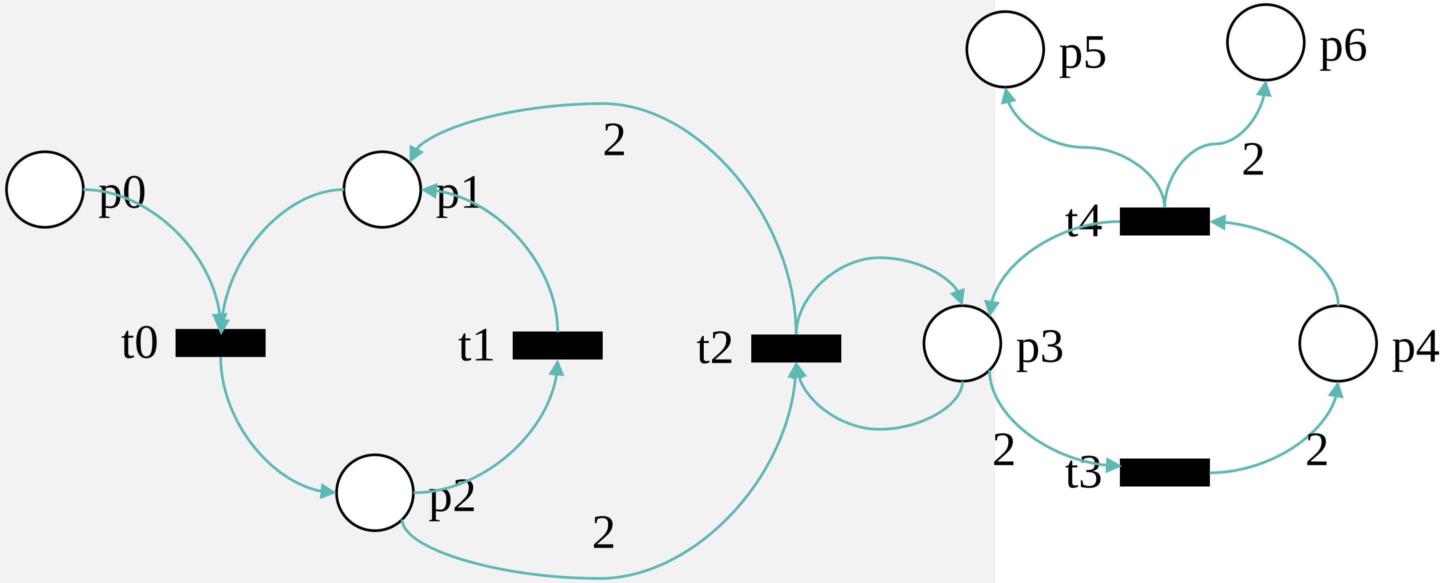


- Step 4 :
  - If no row « r » with single positive or negative entry « k », choose an arbitrary non zero entry
  - For every other column « j » which is non zero on this row
    - Compute coefficient : g=gcd(M[k,r],M[j,r])
    - Add ~(-g or g) times column k to column j to empty cell M[j,r]
    - Update pm (sparsely)
  - Clear column k (sparse)

COMPUTING INVARIANTS

	p0	p1	p2	p3	p4	p5	p6
p0							
p1		1					
p2		1					
p3					1		
p4					1		
p5							-2
p6							1

p1+p2   p3+p4   p6-2*p5

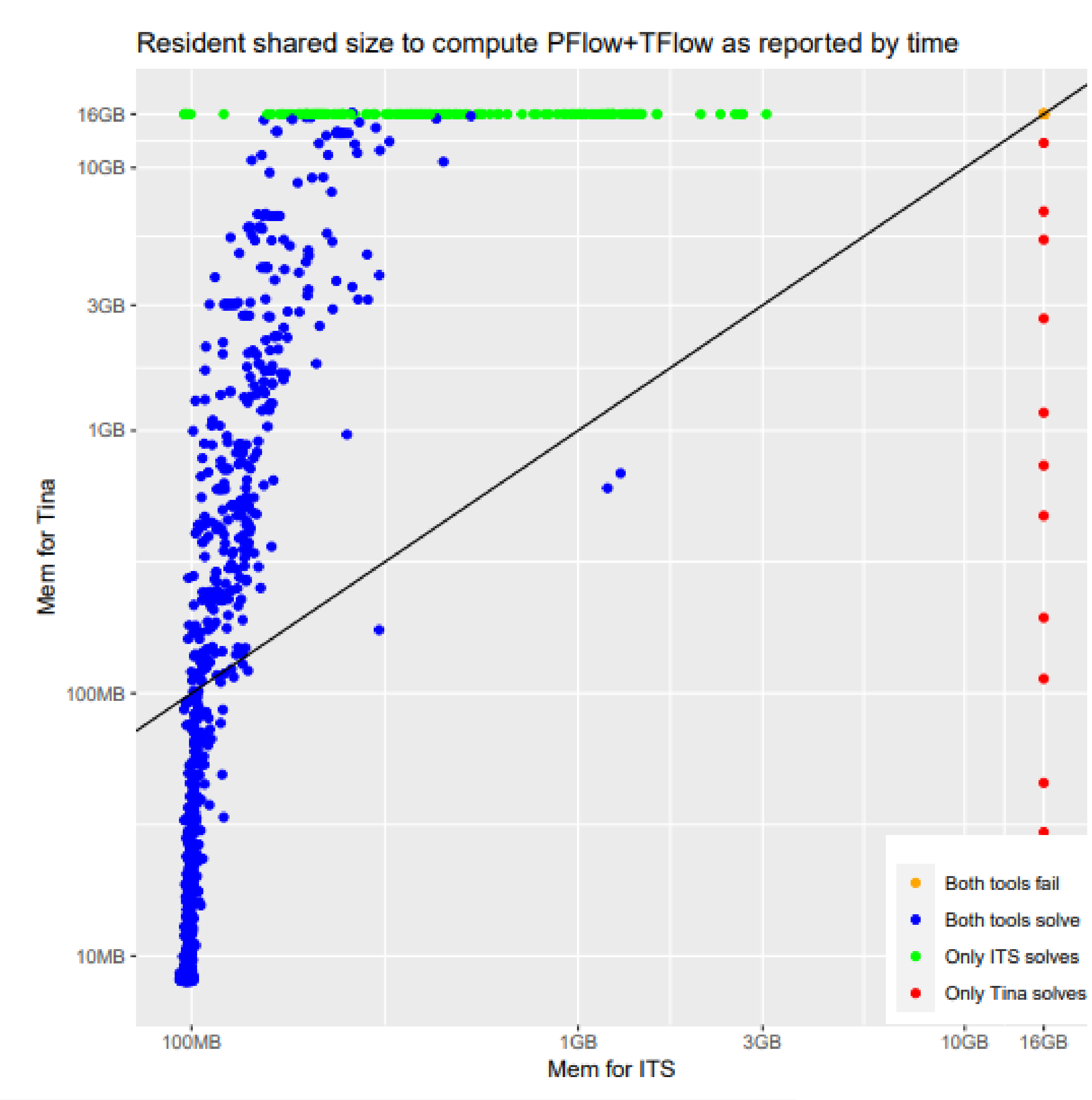


- Step 5 :
  - When matrix is empty, interpret « identity » matrix as invariants
  - For P-invariants, deduce constant value from initial marking
- T-invariants can be computed in the same way, starting from a transposed matrix.

More details and heuristics for choosing a pivot in the paper.

INVARIANTS CONCLUSIONS

- A classical subject revisited
  - Based on PIPE algorithm, derived from d'Anna & Trigila '88 paper
  - Emphasis on **sparse** data structures and operations (**<u>all of them</u>**) to remain in both memory AND time complexity related to non zero entries
  - Implementation in plain Java, no libraries
    - Hacked version of google.android.SparseArray
    - Based on code from APT->PIPE
    - Refined/Profiled implementation (60+ commit on main file over 5 years)
  - Strong constraints, very cheap to compute, very cheap to use e.g. to feed a SMT/ILP solver.
- Very favorable performance comparison to Tina (« struct » component) even when it uses « 4ti2 » library as solver



https://github.com/yanntm/InvariantPerformance

COMPUTING UPPER BOUNDS

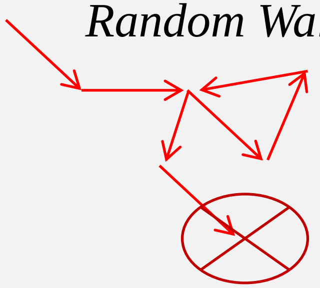
- Problem Statement :
  - Given a set of places, what is the maximum value of their marking in any reachable state ?
- Strategy is based on : Min <= Bound <= Max
  - A Structural Upper Bound : Max
    - We guarantee this bound cannot be exceeded
    - Initialize with +infnity
  - A Reachable Lower Bound : Min
    - We guarantee the bound is not lower than this value
    - Initialize with value of expression in initial marking
- Iteratively try to :
  - Reduce Max
  - Increase Min
- When Min=Max, this is the bound we were looking for.

MIXING DIRECTED WALKS, SMT AND STRUCTURAL REDUCTIONS

PN2020 : Structural Reductions Revisited

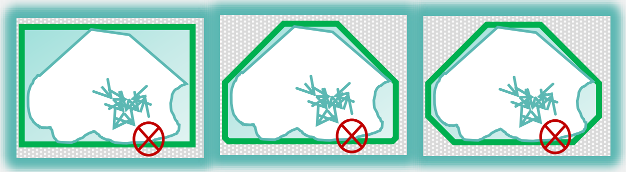
net and property

Random Walk



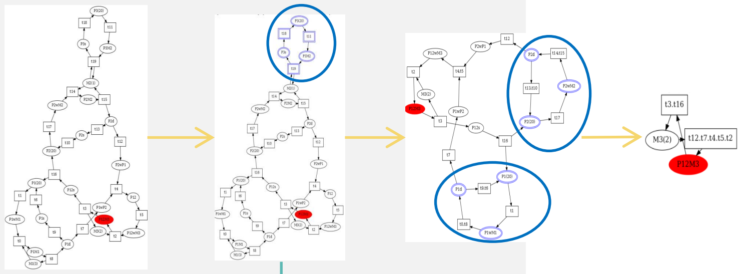
Not found

SMT overapproximation



SAT+ Failed *guided* walk

Structural Reduction



Counter-example

UNSAT

convergence

FALSE
Invariant does not hold

TRUE
Invariant holds

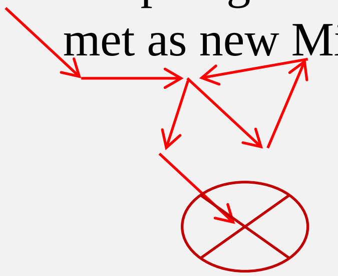
A simpler net and property

MIXING DIRECTED WALKS, SMT AND STRUCTURAL REDUCTIONS

Adapting « PN2020: Structural Reductions Revisited » to Bounds

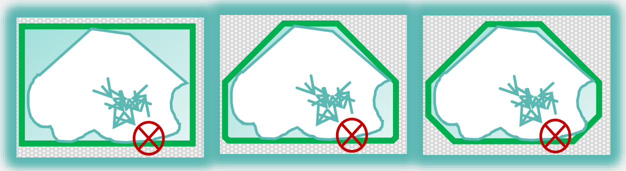
net and expression « e »
Set Max=+inf and Min=m0(e)

Random Walk
Keep largest value met as new Min



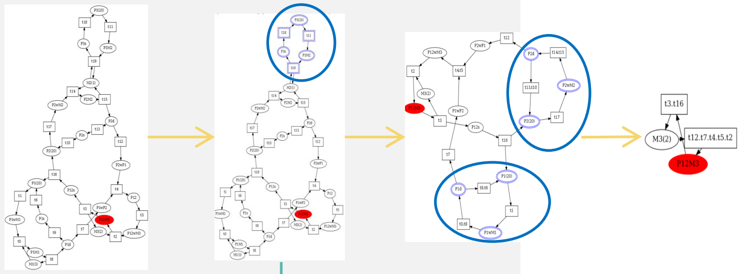
Else
Try to prove Min+1 unfeasible

SMT overapproximation



SAT+ Maximize =>*guided* walk

Structural Reduction



convergence

Max reached
So Min=Max

Conclude

UNSAT
So Min=Max

Conclude

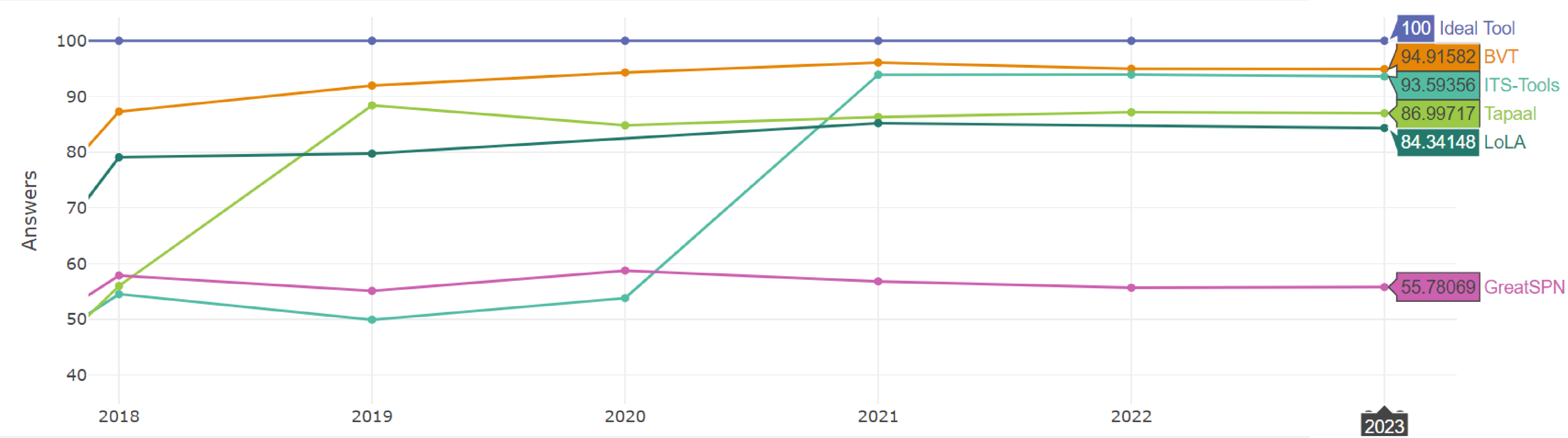
A simpler net and property
Try exhaustive methods/other tools

ADDITIONAL ELEMENTS IN THE WORKFLOW

- For colored nets, use the skeleton first to approximate Max
- Use P-flows (invariants)
  - Positive P-semi-flows : p1 + 2*p2 = 3
    - Max(p1)=3, Max(p2)=1
  - Given some of these constraints use generalized invariants : p0 – p2 = 1
    - We know Max(p2)=1, so Max(p0)=2
  - Provides a very rough Max on expressions, but is very fast and scales well
- Then iterate the modified invariants procedure
  - Random walk to increase Min
  - SMT contraints to test Max <= Min
    - If SAT additionally Maximize expression
    - Try to replay SAT model (using Parikh counts of the solution)
- Also try some exhaustive methods
  - Based on Hierarchical Set Decision Diagrams (SDD)
  - Based on LTSMin + POR

CONCLUSION UPPER BOUNDS

- An iterative refinement that uses :
  - An under approximation given by exploration
  - An over-approximation given by a mix of invariants and more complex SMT constraints
    - Leverage the « maximize » option of Z3 on SAT
  - Structural reductions to study much smaller nets

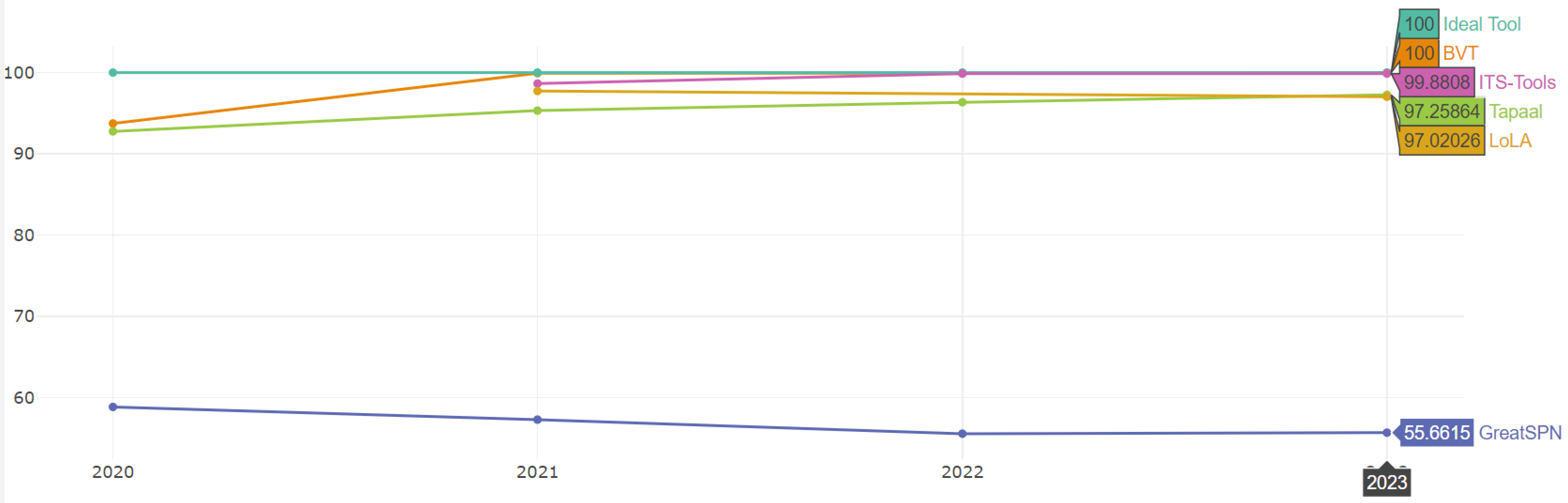


https://yanntm.github.io/MCC-analysis/upper_bounds_annual.html

ONE SAFE, STABLE MARKING

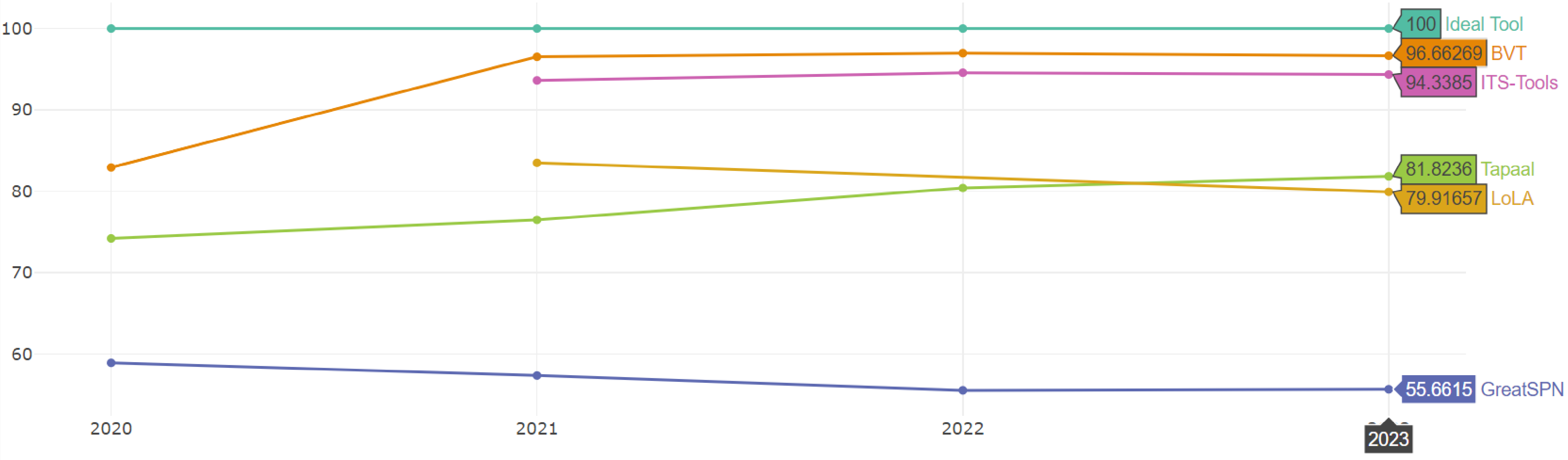
- The paper also presents our strategies for deciding « One Safe » and « Stable Marking »
  - In both cases, prior to using the reachability engine we try to use very simple strategies first
    - Some property specific structural reductions are introduced (e.g. trivial token flow graph)
    - Split into many sub problems (one per place) but try to reduce the number of queries
  - SMT does not really scale up to 10^6 elements, but for larger models we can still try
    - sparse invariant computation,
    - memoryless directed/pseudo-random reachability,
    - structural reductions
  - The process also builds simpler models we can submit to other tools (+red)
  - ITS-Tools won Gold in 2023 at MCC in these examinations
    - All the variants of other tools (+red) that use our engine as pre-treatment also did better than silver medalist Tapaal

ONE SAFE



https://yanntm.github.io/MCC-analysis/global_properties_annual.html

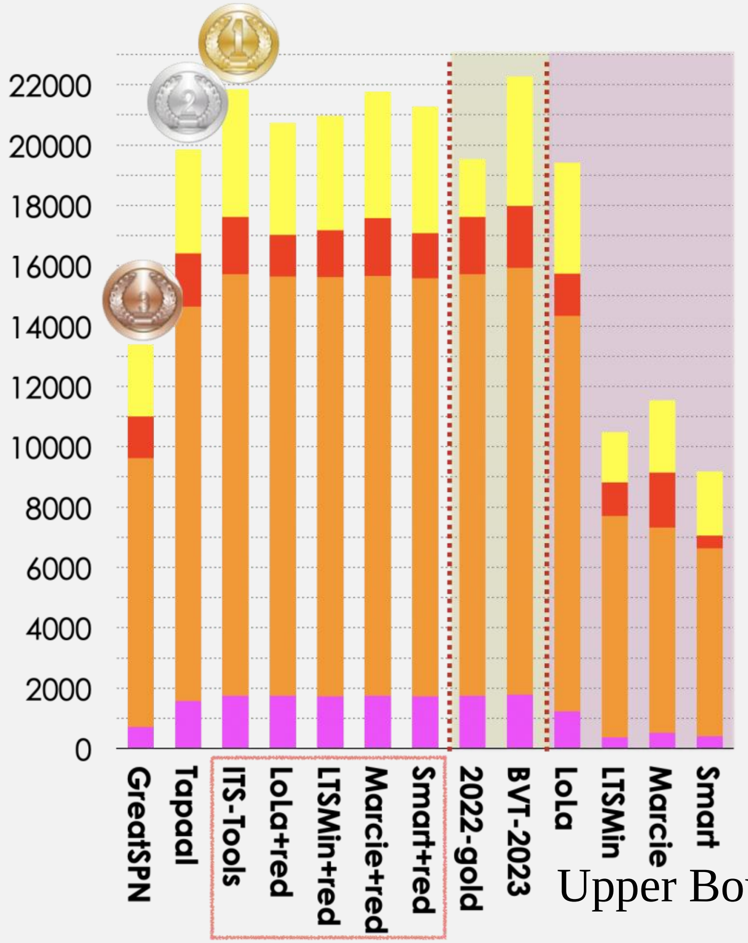
STABLE MARKING



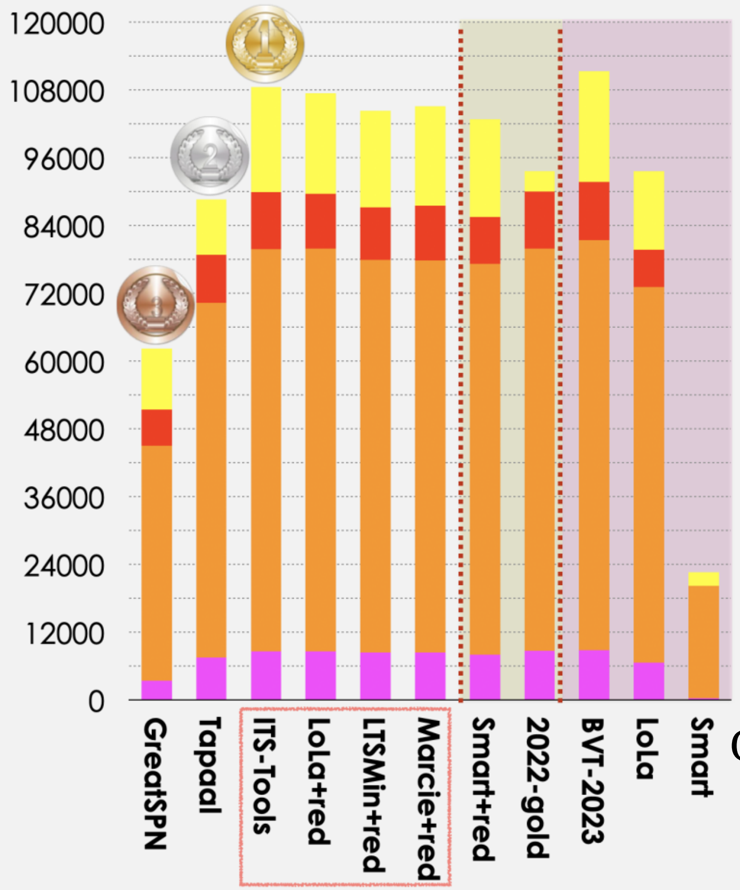
https://yanntm.github.io/MCC-analysis/global_properties_annual.html

CONCLUSION

- The strategies (informally) described in this paper are to the best of our knowledge state of the art solutions to these queries, so we hope this presentation is still useful
- Further directions include linear inequalities, infinite bounds
- All source code fully available as FOSS under GPL on https://github.com/ITSTools



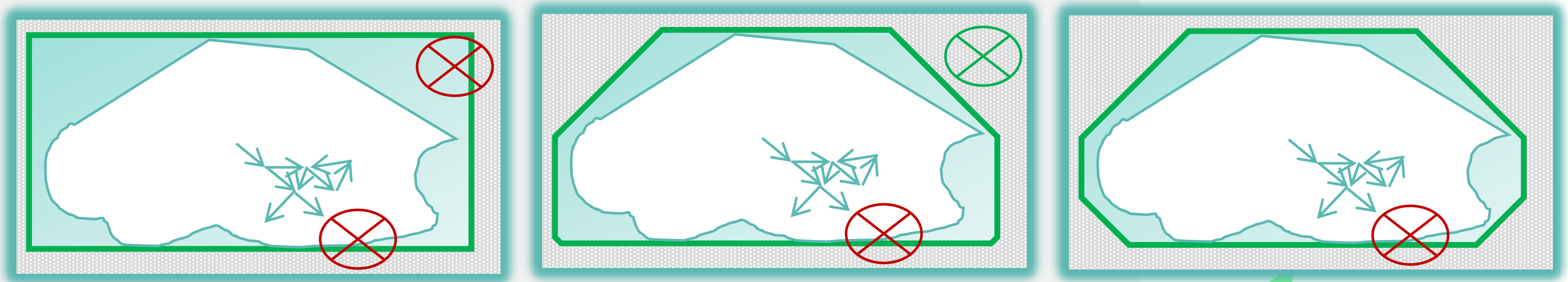
Upper Bounds in MCC'23



Global Properties in MCC'23

SMT CONSTRAINTS

Highlights



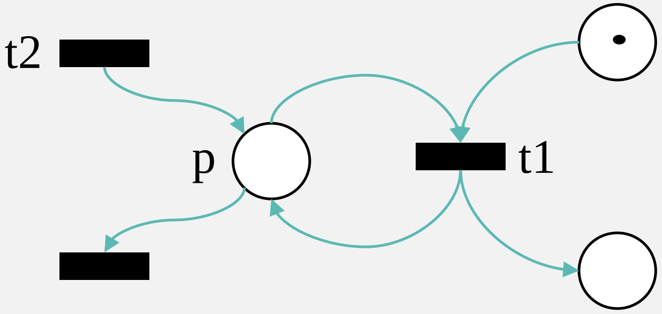
Iterative refinement of the over approximation

- Places = variables
  - P1 >= 0, P2 >= 0…
- Generalized flows
  - P1 + 2*P2 – P3 = 1
- Trap constraints
  - P1 > 0 OR P2 > 0
  - Compute ***useful constraints*** as separate SMT problem
- State Equation
  - Add a positive variable for firing count of transitions
  - P1 = T1 – T2 + 1
- Read => Feed
  - T1 reads P; m0(P)=0 ; T2 and T3 feed P
  - T1 > 0 => T2 > 0 OR T3 > 0
- Causal constraints (*precedes* is a strict partial order)
  - T1 consumes from P ; m0(P)=0 ; T2 and T3 feed P
  - T1 > 0 => (T2>0 AND T2 *precedes* T1) OR (T3 >0 AND T3 *precedes* T1)
  - Is inconsistent (UNSAT) if we also have « T1 *precedes* T2 » and « T1 *precedes* T3 »

+Incremental constraints
+Use Reals then Integers
+UNSAT = invariant proved true
+SAT = candidate state + firing count

READ => FEED

Constraining the transition firing count



- The state equation ignores read arcs

  ⇒ spurious solutions, **t1** and **t2** *are not correlated* in the state equation constraints

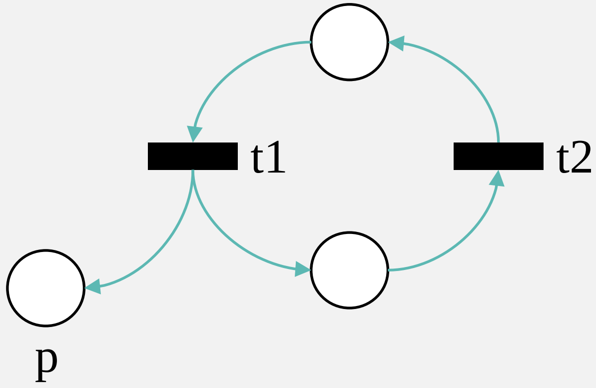
Reason on first occurrence of each transition :

- If a transition has positive firing count and reads in place « p » initially empty, it must be the case that a transition feeding « p » also has positive firing count.

  t1 > 0 => t2 > 0

CAUSAL CONSTRAINTS (UNSAT)

A partial order on first occurrence of each transition



The state equation can borrow non existing tokens

⇒ t1=1 and t2=1 is a solution to the state equation to mark « p »

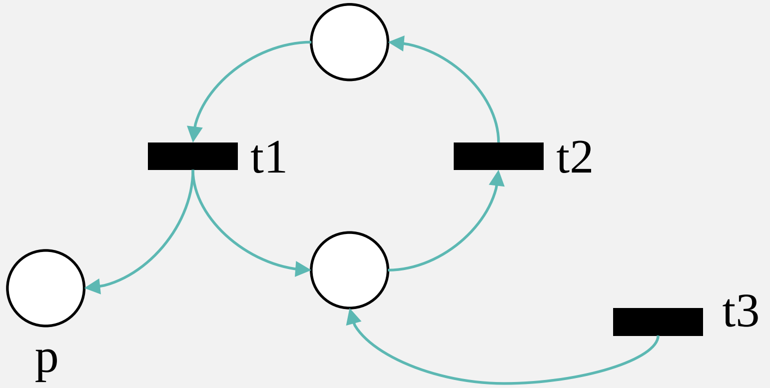
We assert that :

- t1 > 0 => t2 > 0 and t2 precedes t1
- t2 > 0 => t1 > 0 and t1 precedes t2

Obtaining a contradiction (UNSAT) as soon as t1 or t2 positive in the solution

CAUSAL CONSTRAINTS (SAT)

A partial order on first occurrence of each transition



The state equation can borrow non existing tokens

$\Rightarrow$ t1=1 and t2=1 is a solution to the state equation to mark « p »

We assert that :

- t1 > 0 => t2 > 0 and t2 precedes t1
- t2 > 0 => (t1 > 0 and t1 precedes t2) OR (t3 > 0 and t3 precedes t2)

Obtaining a solution (SAT) : t3 precedes t2 precedes t1