

Efficient Algorithms for Three Reachability Problems in Safe Petri Nets

Pierre Bouvier Hubert Garavel

Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, France



Three reachability problems

- We focus on ordinary safe Petri Nets

- **Dead Places Problem:**

- ▶ a **place** is **dead** if it never gets a token
- ▶ for each place p , decide $\neg R(\{p\})$, where
 $R(M) \stackrel{\text{def}}{=} \text{there exists a reachable marking in which } M \text{ is included}$

- **Dead Transitions Problem:**

- ▶ a **transition** is **dead** if it is never enabled
- ▶ for each transition t , decide $\neg R(\bullet t)$

- **Concurrent Places Problem:**

- ▶ **two places** are **concurrent** if they can have a token simultaneously
- ▶ for each two places p_1 and p_2 , decide $R(\{p_1, p_2\})$
- ▶ concurrency between places is symmetric and quasi-reflexive

Why are these problems interesting?

■ Dead places and dead transitions:

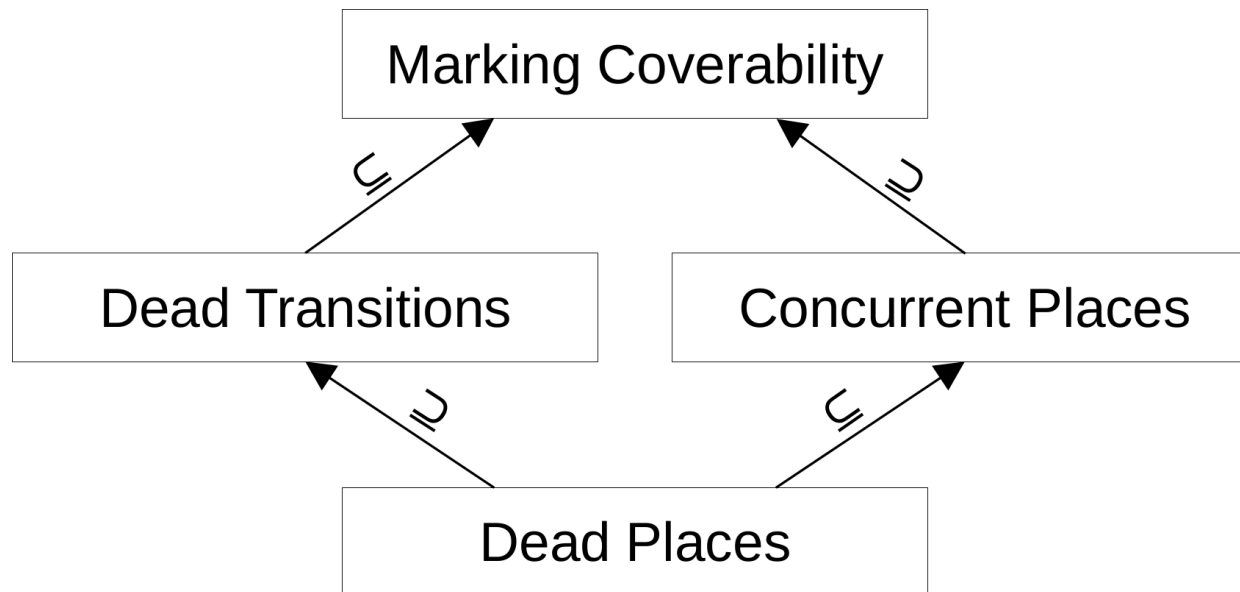
- ▶ useful for **simplifying complex Petri nets**, especially those generated from higher-level formalisms
- ▶ profitable reduction: **20.4% dead places** and **37.7% dead transitions** in practice

■ Concurrent places:

- ▶ crucial role for the **decomposition of Petri nets** into automata networks [Bouvier et al., Petri Nets 2020]
- ▶ statistically: **67% non-concurrent pairs of places**

Complexity of these problems

- 3 subproblems of the Marking Coverability Problem:



- These 3 problems are **PSPACE-complete**

What about non-safe Petri nets?

- Concurrent places are most interesting on safe Petri nets
- For any state machine having no dead place:
 - ▶ 1 initial token
 $\Rightarrow$ each place is only concurrent with itself
 - ▶ 2 initial tokens
 $\Rightarrow$ all places are pairwise concurrent

Practical motivation

- Despite PSPACE complexity, we seek for efficient algorithms that solve a majority of problems
- Benchmarks:
 - ▶ we use a collection of 13,116 nets from **academia**, **industry**, and **competitions**
 - ▶ these models are **diverse** and **complex**

property	yes	no	property	yes	no
pure	62.9%	37.1%	connected	94.0%	6.0%
free choice	41.3%	58.7%	strongly connected	14.3%	85.7%
extended free choice	42.7%	57.3%	conservative	16.5%	83.5%
marked graph	3.5%	96.5%	sub-conservative	29.7%	70.3%
state machine	12.1%	87.9%	non trivial and unit safe	67.7%	32.3%

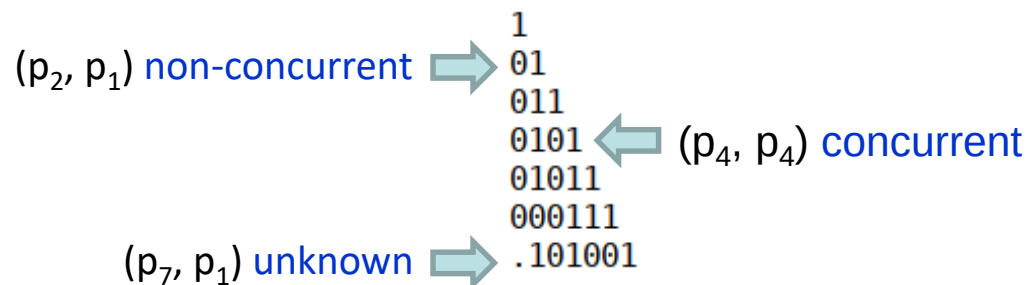
feature	min value	max value	average	median	std deviation
#places	1	131,216	282.4	15	2690
#transitions	0	16,967,720	9232.8	20	270,287
#arcs	0	146,528,584	72,848	55	2,141,591
arc density	0.0%	100.0%	14.5%	9.4%	0.2

Straightforward approach

- Reuse **existing model checkers** for Petri nets:
 - ▶ encode the 3 problems as temporal-logic formulas
 - ▶ analyse model-checking results to get dead places, dead transitions, and concurrent places
- Possible, yet **inefficient**:
 - ▶ linear or quadratic number of formulas
(300 places $\Rightarrow$ 45,150 formulas for concurrent places)
 - ▶ redundant calculations: many similar formulas need to be evaluated on the same Petri net

Dedicated approach

- Instead, we suggest tools with built-in options:
 - ▶ option **-dead-places**
result = vector of {dead, non-dead, unknown} values
indexed by place numbers
 - ▶ option **-dead-transitions**
result = vector of {dead, non-dead, unknown} values
indexed by transition numbers
 - ▶ option **-concurrent-places**
result = half-matrix of {concurrent, non-concurrent, unknown}
values, indexed by place numbers



Algorithms for computing the vectors of dead places and dead transitions

1. Marking graph exploration

- Explore all reachable markings, e.g., using decision diagrams
 - ▶ PSPACE-complete $\Rightarrow$ may take too long or too much memory
- Algorithmic enhancements:
 - ▶ **timeout** or limit on exploration depth
 - ▶ **speed-up** calculations by not firing already known dead transitions
 - ▶ **shortcuts**: halt exploration as soon as all results are known
- Expected results, for all places and transitions:
 - ▶ **complete exploration**: gives **dead** or **non-dead** values
 - ▶ **incomplete exploration**: gives **non-dead** or **unknown** values
in this case, we apply additional algorithms to remove as many **unknown** values as possible

2. Structural rules

■ 8 simple theorems to compute some dead or non-dead values:

- Any place belonging to the initial marking M_0 is not dead.
- Any transition having no input place (and no output place) is not dead.
- If a place p is dead, all the transitions of ${}^\bullet p \cup p^\bullet$ are also dead.
- If a transition t is not dead, all the places of ${}^\bullet t \cup t^\bullet$ are also not dead.
- If a transition t is dead, any place p such that ${}^\bullet t = \{p\}$ is also dead.
- If a place p is not dead, any transition t such that ${}^\bullet t = \{p\}$ is also not dead.
- If the net is safe, any transition whose input places form a strict subset of the output places is dead.
- If the net is unit safe, any transition having at least two input (resp. two output) places located in two non-disjoint NUPN units is dead.

■ These rules are applied repeatedly until saturation

3. Linear over-approximation

■ Abstraction:

- ▶ the set of reachable markings is replaced by a set E of places such that, at the end of the algorithm:
 - $p \notin E \Rightarrow$ place p is dead
 - $\bullet t \not\subseteq E \Rightarrow$ transition t is dead

■ Algorithm:

- ▶ initially, E is the initial marking
- ▶ repeat until saturation: for each t , $\bullet t \subseteq E \Rightarrow t \bullet \subseteq E$

- This gives, for each place and transition, either a dead or an unknown value

Combination of approaches

- Approaches 1-3 are combined in a **well-chosen order**:
 - ▶ structural rules
 - ▶ linear over-approximation
 - ▶ marking graph exploration
 - ▶ structural rules (again)
- Two implementations:
 - ▶ **Caesar.bdd**: 11K lines of C code (using Cudd for BDDs)
 - ▶ **ConcNUPN**: 730 lines of Python

ConcNUPN is used to cross-check results of Caesar.bdd

Experimental results using Caesar.bdd

problem	value of t	0	5	10	15	30	45	60	120	180	240	300
dead places	% complete vectors	44.6	93.0	93.6	93.8	94.4	94.6	95.1	95.3	95.4	95.5	95.6
	% unknowns values	48.9	33.5	32.0	31.3	28.9	28.3	27.9	27.1	26.5	25.9	25.8
	% vector completion	69.3	97.0	97.3	97.5	97.7	97.9	97.9	98.1	98.1	98.2	98.2
dead trans.	% complete vectors	29.3	92.3	92.9	93.2	93.7	94.0	94.1	94.4	94.7	94.9	95.0
	% unknowns values	68.7	65.0	63.5	62.0	61.0	59.3	57.8	54.6	45.2	39.9	29.8
	% vector completion	50.9	95.8	96.2	96.4	96.7	96.8	96.9	97.1	97.2	97.3	97.3

■ Dead places (with a BDD timeout of 60 s):

- ▶ fully solved vectors (no unknown values): 95.1%
- ▶ average number of unknown values in vectors: 2.1%

■ Dead transitions (with a BDD timeout of 60 s):

- ▶ fully solved vectors (no unknown values): 94.1%
- ▶ average number of unknown values in vectors: 3.1%

Algorithms for computing the half matrix of concurrent places

1. Marking graph exploration

- First, explore reachable markings, e.g., using decision diagrams:
 - ▶ PSPACE-complete: the exploration may be **incomplete** (**timeout** or limit on exploration depth)
 - ▶ **shortcuts** are impossible or very unlikely, contrary to the marking graph exploration for dead places
- Then, check for all pairs of places whether it exists a reachable marking containing these places
- Expected results for all pairs of places:
 - ▶ **complete exploration**:
gives **concurrent** or **non-concurrent** values
 - ▶ **incomplete exploration**:
gives **concurrent** or **unknown** values

2. Structural rules

■ 8 theorems giving concurrent or non-concurrent pairs:

- The places of the initial marking M_0 are pairwise concurrent.
- If a transition is not dead, its input places (resp. output places) are pairwise concurrent.
- A non dead place is concurrent with itself.
- A dead place is non concurrent with any other place, including itself.
- If a dead transition has two (distinct) input places, these places are non concurrent.
- If a transition t (dead or not) has a single input place p , this place is non concurrent with any output place of t different from p .
- For any path $(p_1, t_1, p_2, t_2, \dots, p_n, t_n, p_{n+1})$ such that each transition t_i has a single input place p_i and at least one output place p_{i+1} , the places p_1 and p_{n+1} are non concurrent if they are distinct.
- If the net is a unit-safe NUPN, any two distinct places located in non-disjoint units are non concurrent. In particular, any two distinct places located in the same unit are non concurrent.

■ These rules are applied until saturation, together with theorems for dead places and dead transitions

3. Quadratic under-approximation

- Abstraction: the set of reachable markings replaced by a set E of pairs of places such that $\{p_1, p_2\} \in E \Rightarrow p_1 \text{ and } p_2 \text{ concurrent}$
- 4 theorems repeated until saturation:
 - If a place p is not dead, any transition t such that $\bullet t = \{p\}$ is also not dead.
 - If two (distinct) places p_1 and p_2 are concurrent, any transition t such that $\bullet t = \{p_1, p_2\}$ is not dead.
 - If a transition is not dead, its output places are pairwise concurrent.
 - If two distinct places p_1 and p_2 are concurrent, p_2 is also concurrent with each output place of any transition t such that $\bullet t = \{p_1\}$.
- This gives certain pairs of concurrent places

4. Quadratic over-approximation

- Generalizes a former algorithm [Kovalyov & Esparza, 1996]
- Abstraction: the set of reachable markings is replaced by a set E of pairs of places, such that, at the end of the algorithm: p_1 and p_2 concurrent $\Rightarrow \{p_1, p_2\} \in E$
- Algorithm:
 - ▶ operator $M_1 \otimes M_2 \stackrel{\text{def}}{=} \{\{p_1, p_2\} \mid p_1 \in M_1 \wedge p_2 \in M_2\}$
 - ▶ auto-product $M^{(2)} \stackrel{\text{def}}{=} M \otimes M$
 - ▶ initially $E = M_0^{(2)}$, where M_0 is the initial marking
 - ▶ repeat until saturation: for each transition t ,
for each set of places M : $\bullet t \subseteq M \wedge M^{(2)} \subseteq E \Rightarrow ((M \setminus \bullet t) \cup t\bullet)^{(2)} \subseteq E$
- This gives, for each pair of places, either a non-concurrent or an unknown value

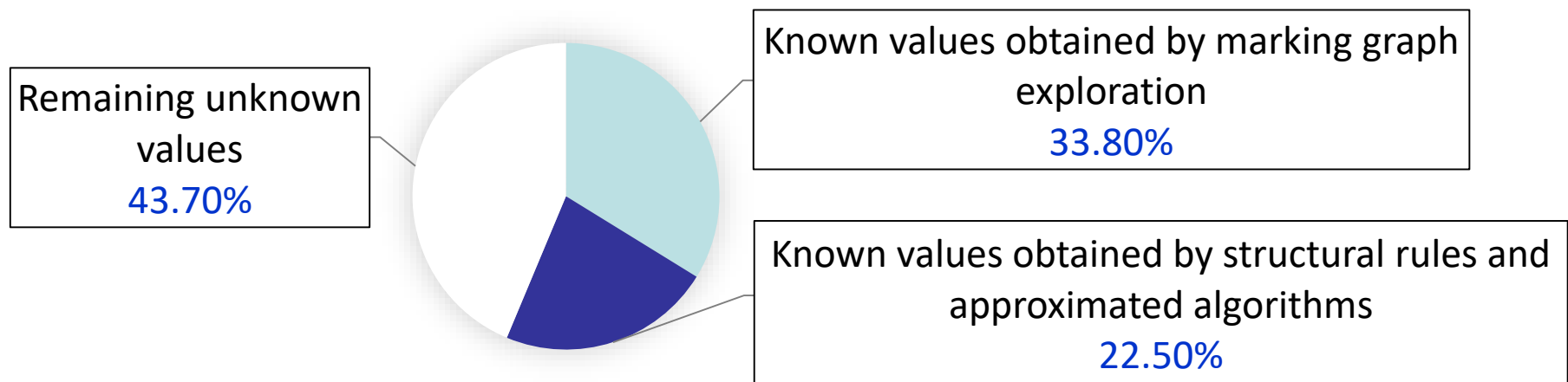
Combination of approaches

- Approaches 1-4 are combined in the following order:
 - ▶ marking graph exploration
 - ▶ structural rules
 - ▶ quadratic under-approximation
 - ▶ quadratic over-approximation
- Implemented in Caesar.bdd and ConcNUPN

Experimental results using Caesar.bdd

value of t	0	5	10	15	30	45	60	120	180	240	300	360	420
% complete matrices	51.0	91.6	92.2	92.5	93.0	93.6	94.0	94.2	94.4	94.5	94.6	94.7	94.7
% unknowns values	45.0	44.7	44.7	44.4	44.4	43.7	43.7	43.7	43.6	43.6	43.6	43.6	43.6
% matrix completion	81.6	96.3	96.6	96.8	97.0	97.1	97.2	97.3	97.4	97.4	97.4	97.5	97.5

- For a BDD timeout of 60 seconds:
 - ▶ fully solved matrices (no unknown values): 94%
 - ▶ average number of unknown values in matrices: 2.8%
- For the few incomplete half matrices:



Conclusion

Conclusion

- Three **useful**, yet **difficult problems** (PSPACE-complete)
- Combination of approaches to handle large models:
 - ▶ $\approx 95\%$ of models are completely solved (on **13,000+** nets)
 - ▶ some large models are **partially** solved
(the solution contains unknown values)
- Future work: **remove more unknown values**
using, e.g., invariants, partial orders, SAT solving,
structural reductions, etc.
- **Other tools** are starting to address these problems:
 - ▶ **Kong** (Nicolas Amat, LAAS-CNRS) – structural reductions
 - ▶ **ITS-Tools** (Yann Thierry-Mieg, LIP6) – model checking