Summary

Complex distributed systems often require correctness in safety critical environments, which can be achieved by using mathematical formalisms to model the behaviour and verify properties using computer aided model checking. One such formalism is colored Petri nets, which is well-suited for asynchronous systems and has a pre-existing ecosystem of software tools.

The state of the art model checkers convert colored Petri nets into equivalent Petri nets to re-use existing methods and techniques. We propose and explore an approach where queries are checked directly by exploring the state space of the colored Petri net. We previously implemented an algorithm and search techniques to achieve this for reachability queries and proved them correct with experiments showing promising results. In this thesis we extend our approach to also include LTL queries as well as introduce two new techniques to solve queries more efficiently: query simplification via over-approximation and using color constraints when generating bindings. We prove the new techniques correct and run experiments to test their effectiveness.

In our experiments we compare our approach with the unfolding approach used in the tool TAPAAL on queries from the Model Checking Competition 2024. Compared to the previous implementation we have a significant increase in the amount of queries solved without a counter example due to our new techniques. For colored Petri nets our approach now overall outperforms TAPAAL on both cardinality- and fireability-queries, with a significant improvement to fireability queries. For LTL queries our technique does not perform well overall and is significantly behind TAPAAL but still provides 370 unique answers compared to TAPAAL's combined answers using Tarjan and NDFS. We also experimentally test combining our approach and TAPAAL's unfolding in the script used by TAPAAL in the Model Checking Competition 2025 for reachability queries where on our dataset 11% more queries were answered.

Improvements in Explicit State Space Exploration for Colored Petri Nets

Emil Normann Brandt, Jens Emil Fink Højriis, and Kira Stæhr Pedersen

Aalborg University
Department of Computer Science, Aalborg, Denmark

Abstract. In model checking of colored Petri nets, the state of the art approach is to unfold colored nets into equivalent Petri nets. This allows the reuse of existing Petri net techniques but also adds additional overhead which in some cases is so large that unfolding within reasonable time is infeasible. Instead using an explicit state space exploration approach has shown promising results for reachability queries. In this thesis we extend the approach to include LTL queries as well as propose new optimisations and techniques, namely query simplification by over-approximation and using color constraints derived from the current state of the net when generating bindings. We implement these optimisations as an addition to the tool TAPAAL and experimentally test the performance on a dataset from the 2024 Model Checking Competition, an annual contest for tools in verifying properties for Petri nets. The experimental results indicate that both techniques are effective, as well as helping to solve different queries. The experiments for LTL queries show that our implementation overall performs significantly worse than the current unfolding approach but is able to find some unique answers. In the experiments for reachability queries, our implementation overall outperforms TAPAAL using unfolding answering 9% more queries. The improvement is especially significant for fireability queries as well as queries that can be solved by counter example.

1 Introduction

Asynchronous systems are inherently complex and hard to design correctly. To aid in the design of asynchronous systems, model checking can be used to verify that systems are correct. *Petri nets* (PNs) were first invented by Carl Adam Petri [24] to aid in modelling and verification of asynchronous systems. Many extensions have been proposed to make PNs either more expressive, easier to create, or easier to visually understand. One popular extension suggested by Kurt Jensen is *colored Petri nets* (CPNs) [13] which adds colors to tokens. The intention is to make modelling easier in cases where a system has many instances of the same component that need to be modelled, while still being able to distinguish between them. As other types of verification CPNs suffer from the state space explosion problem, especially if large color types are used, since the size of the state space can increase exponentially. A CPN with finite color types has

the same expressiveness as an ordinary PN, which means that a CPN can be unfolded to an equivalent PN [16] and they will both have equally large state spaces. Unfolding is desirable because the existing query answering optimisations for PNs can be applied directly. The downside to unfolding is that both the additional overhead and the resulting PN can be so large that either the unfolding or the resulting net exceeds time- or memory-constraints. This means that some queries that could be answered by exploring very few states cannot be solved.

In [11] we introduced and implemented an algorithm in TAPAAL [8] to solve reachability queries for CPNs using explicit state space exploration. It was experimentally shown that the explicit exploration approach surpassed TAPAAL when solving queries with a counter example and was able to solve many queries not being solved with unfolding. We now extend our explicit exploration to cover LTL queries, as well as propose improvements for reachability queries. Our experiments show that the improvements have a significant impact. Our explicit approach surpasses unfolding for reachability queries without a counter example and extends the gap for queries with a counter example, resulting in the explicit exploration approach overall performing better than unfolding. For LTL queries, although our approach solves significantly fewer queries than TAPAAL using unfolding, it still answers 370 unique queries.

Related Work The state of the art CPN model checkers that competed in the *Model Checking Competition* (MCC)'24 [18], use various techniques to verify properties. ITS-Tools [26] uses skeleton analysis to over-approximate the state space of a CPN and if inconclusive it uses unfolding [1]. Lola [27] uses explicit state space exploration for PNs and supports CPNs by unfolding [1]. TAPAAL uses a mix of different explicit approaches, but always unfolds CPNs to PNs, before doing any explicit state space exploration [1]. To our knowledge, no state of the art tool which competed in MCC'24 currently uses explicit CPN state space exploration for verifying properties.

All tools use Büchi automata [19,25] to verify LTL queries for PNs[14,15]. Multiple algorithms exist for LTL such as *Nested Depth-First Search* (NDFS) [6] and Tarjan's algorithm [10].

In [11] we designed a successor generator that iterated through bindings relatively naively, with the primary optimisations being not generating bindings when the amount of tokens in a place was too low for any binding to be able to enable the transition, and only iterating through bindings where relevant variables are changed.

Different forms of query simplification and reduction for uncolored nets are used by many tools, such as detecting tautologies and using linear programming [1,4] allowing some queries to either be solved or reduced in size without exploring any states. These techniques if naively reused for colored nets require unfolding the net first, causing added overhead and complexity issues. Color over-approximation is a query simplification technique for colored nets implemented in [17], the technique removes colors and guards and uses the result of

the simplified query if applicable. In [17] the technique discards queries involving negated fireability expressions or arc expressions with subtraction that are not trivially equivalent to an expression without subtraction, where our approach further analyses the query to determine whether it can be simplified.

Bibliographic remarks This paper is a continuation of [11], therefore we reuse the same basic semantics for CPNs, specifically:

- Section 1 and Related Work both reuse the first paragraph with minor changes to improve readability.
- Section 2 reuses the first two paragraphs with minor changes to improve readability and with the addition of A^ω and a new symbol $\mathcal{B}$ to denote multisets.
- Section 2.1 is reused with minor changes to improve readability with the addition of definitions for *max-cardinality* as well as pre- and post-places.
- Figure 1 uses the same example net, but Figure 1b has a different layout.
- Section 2.4 is reused.
- Section 2.5 is reused except an example has been removed and the last paragraph has been shortened.

2 Preliminaries

Let $\mathbb{N}$ be the set of positive integers, $\mathbb{N}^0 = \mathbb{N} \cup \{0\}$ and $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$. A multiset over A is a function $S : A \to \mathbb{N}^0$. A possible multiset for $A = \{a, b\}$ can be denoted as $S = 1'a + 2'b$ which means $S(a) = 1$ and $S(b) = 2$. Let $\mathcal{B}(A)$ be the set of all multisets over A. The standard multiset operations $\in$, $\subseteq$, $\uplus$ and $\setminus$ are assumed to be defined. The cardinality operator is defined as $|S| = \sum_{a \in A} S(a)$. Let A^* be the set of all sequences, A^+ the set of non-empty sequences and A^ω the set of all infinite sequences of elements in A. We write sequences as either abc or (a, b, c).

2.1 Colored Petri Net

Let BCT be a finite set of basic color types and $\mathbb{C}$ be a finite set of colors. Let $colors$: $BCT \to 2^{\mathbb{C}} \setminus \emptyset$ be a function, which given a basic color type returns a non-empty set of colors. Each color type in BCT has unique colors s.t. if $colors(bct) \cap colors(bct') \neq \emptyset$ then $bct = bct'$ for all $bct, bct' \in BCT$.

We assume that the set returned by $colors$ has an implicit ordering ($<$,$=$ and $>$). We additionally assume that the elements have successor and predecessor functions available working cyclically that can be invoked with $++$ and $--$ respectively.

Example 1. If we have the set of colors $\{a, b, c\}$ then $a < b$, $b-- = a$, $c++ = a$ and $c++ < c$.

The set of color types CT is a finite set where $BCT \subseteq CT$ and $\mathbb{C} \cap CT = \emptyset$. Let $type : CT \to BCT^+$ return a non-empty sequence of the composite color types. If $type(ct) = type(ct')$ then $ct = ct'$ and if $|type(ct)| = 1$ then $type(ct) = ct$ and $ct \in BCT$.

Let $colors : CT \to 2^{\mathbb{C}^+}$, return the set of all possible color sequences for a color type s.t. $colors(ct) = \{(c_1, \ldots, c_n) \mid c_i \in colors(bct_i) \text{ for all } i \in \mathbb{N}\}$ where $type(ct) = bct_1 \ldots bct_n$.

Let $\mathcal{CB}(ct) = \mathcal{B}(colors(ct))$ be the set of all multisets over $colors(ct)$.

Example 2. Starting a running example, let $\mathbb{C} = \{a, b, c\}$, $BCT = \{base\}$, $CT = \{base, product\}$, $type(base) = (base)$, $type(product) = (base, base)$ and $colors(base) = \{a, b, c\}$ then $5'(a) + 2'(b)$ and $1'(b, a) + 2'(a, b) + 3'(c, c)$ are possible multisets for *base* and *product*, respectively.

Let $\mathbb{V}$ be a finite set of variables and let $type : \mathbb{V} \to BCT$ be a function that maps a variable to a basic color type. We reuse *type* as a function name, since the signature makes it clear what function is being used. Let $binding : \mathbb{V} \to \mathbb{C}$ be a function that maps a variable to a color s.t. if $binding(v) = c$ then $c \in colors(type(v))$ where $v \in \mathbb{V}$ and $c \in \mathbb{C}$. A binding associates each variable with colors of its color type. The set of all bindings is denoted as $\mathbb{B}$ and by convention we define $\mathbb{B} = \{\perp\}$ iff $|\mathbb{V}| = 0$.

Example 3. Continuing the running example where $\mathbb{C} = \{a, b, c\}$, let $\mathbb{V} = \{x, y\}$, $type(x) = type(y) = base$ then $\mathbb{B} = \{[x \mapsto a, y \mapsto a], [x \mapsto a, y \mapsto b], [x \mapsto a, y \mapsto c], ...\}$ s.t. $|\mathbb{B}| = 9$ which means there are nine different possible bindings.

An arc expression of color type $ct \in CT$ is defined as:

$$
\begin{aligned}
ae_{ct} &::= m * ae_{ct} \mid ae_{ct} + ae_{ct} \mid ae_{ct} - ae_{ct} \mid m'\alpha \\
\alpha &::= (\beta_1, \ldots, \beta_n) \\
\beta_i &::= \beta_i{-}{-} \mid \beta_i{+}{+} \mid c_i \mid v_i
\end{aligned}
$$

where $ct \in CT, v_i \in \mathbb{V}, c_i \in \mathbb{C}$, and $m, n, i \in \mathbb{N}$ s.t. $type(ct) = bct_1 ... bct_n$ and $c_i \in colors(bct_i)$ and $type(v_i) = bct_i$ for all $i \leq n$.

We denote the set of all arc expressions of a certain color type as AE_{ct} and the set of all arc expressions as AE. A special arc expression *nil* is included in AE_{ct} for all $ct \in CT$.

The syntax describes that an arc expression consists of an arbitrary amount of expressions where each expression is an instance of a color type multiplied by a factor, each color can be incremented/decremented individually.

Let $variables : AE \to 2^{\mathbb{V}}$ be a function that given an arc expression returns every variable in the expression.

Let $[\![ae]\!]_b$ where $ae \in AE$ and $b \in \mathbb{B}$, be the evaluation of an arc with regards to a specific binding, when evaluating an arc expression each variable is replaced by its binding and results in a multiset. The special arc expression *nil* represents the empty multiset s.t. $[\![nil]\!]_b = \emptyset$ for any $b \in \mathbb{B}$.

Example 4. Continuing our example, let $ae = 5'(x) - 1'(b)$, $ae' = 2'(x, y) + 3'(b, a)$ and $b = [x \mapsto b, y \mapsto a]$ then $[\![ae]\!]_b = 4'(b)$ and $[\![ae']\!]_b = 5'(b, a)$. Notice

that evaluating ae with different bindings can result in multisets with different cardinalities since an element in a multiset cannot have a negative cardinality.

Let $\mathit{min\text{-}cardinality} : AE \to \mathbb{N}^0$ be a function that given an arc expression returns the lowest cardinality it can evaluate to, defined as $\mathit{min\text{-}cardinality}(ae) = \min_{b \in \mathbb{B}} |[\![ae]\!]_b|$ and let $\mathit{max\text{-}cardinality} : AE \to \mathbb{N}^0$ be a function that given an arc expression returns the highest cardinality it can evaluate to, defined as $\mathit{max\text{-}cardinality}(ae) = \max_{b \in \mathbb{B}} |[\![ae]\!]_b|$.

Example 5. Let $ae = 5'(x) - 1'(b)$ and $\mathbb{B} = \{x \mapsto a, x \mapsto b, x \mapsto c\}$ then $\mathit{min\text{-}cardinality}(ae) = min\{|[\![ae]\!]_{x \mapsto a}|, |[\![ae]\!]_{x \mapsto b}|, |[\![ae]\!]_{x \mapsto c}|\} = min\{|5'(a)|, |4'(b)|, |5'(c)|\} = 4$.

A guard expression is described by the following syntax:

$$
\begin{aligned}
g &::= \alpha^{ct} = \alpha^{ct} \mid \alpha^{bct} \bowtie \alpha^{bct} \mid g \vee g \mid g \wedge g \mid \neg g \mid \mathit{true} \\
\alpha^{ct} &::= (\beta_1^{ct}, \ldots, \beta_n^{ct}) \\
\beta_i^{ct} &::= \beta_i^{ct}{-}{-} \mid \beta_i^{ct}{+}{+} \mid c_i^{ct} \mid v_i^{ct}
\end{aligned}
$$

where $\bowtie \in \{<, \leq, >, \geq, =, \neq\}$, $ct \in CT$, $bct \in BCT$, $c_i^{ct} \in \mathbb{C}$, $v_i^{ct} \in \mathbb{V}$ and $i, n \in \mathbb{N}$ s.t. $bct_1 ... bct_n = type(ct)$ and for all $i \leq n$ then $c_i^{ct} \in colors(bct_i)$ and $type(v_i^{ct}) = bct_i$.

Let $\mathcal{G}$ be the set of all guard expressions and $[\![g]\!]_b$ be the evaluation of a guard expression, where $g \in \mathcal{G}$ and $b \in \mathbb{B}$. When evaluated each variable is replaced by its binding and results in a boolean value.

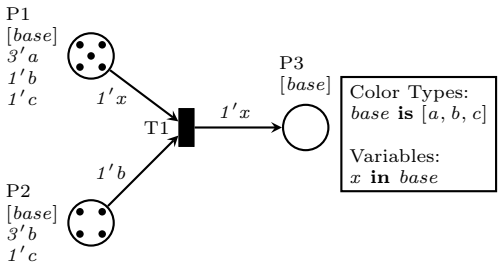
Example 6. If we let $g \equiv a < c \wedge y < b$ and $g' \equiv (a, x) = (a, y) \vee x = a$ and $b = [x \mapsto a, y \mapsto b]$ then $[\![g]\!]_b = false$ and $[\![g']\!]_b = true$.

Let $\mathit{variables} : \mathcal{G} \to 2^{\mathbb{V}}$ be a function that given a guard expression returns every variable in the expression.

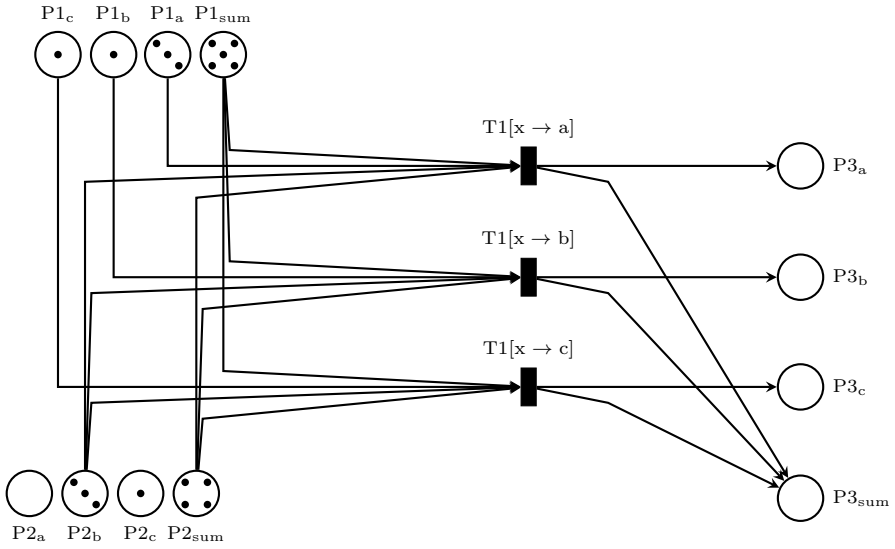
A marking $M : P \to \bigcup_{ct \in CT} \mathcal{CB}(ct)$ is a function describing the state of a CPN.

Definition 1. *A Colored Petri Net is a tuple* $\mathcal{N} = (P, T, CT, \mathbb{V}, C, \mathcal{A}, \mathcal{I}, G, M_0)$, *where*

1. P *is a finite set of places*
2. T *is a finite set of transitions s.t.* $P \cap T = \emptyset$
3. CT *is a finite set of color types*
4. $\mathbb{V}$ *is a finite set of variables*
5. $C : P \to CT$ *is a function from a place to a color type*
6. $\mathcal{A} : (T \times P) \cup (P \times T) \to AE$ *is an arc expression function s.t.* $\mathcal{A}(t, p) \in AE_{C(p)}$ *and* $\mathcal{A}(p, t) \in AE_{C(p)}$ *for all* $p \in P$ *and* $t \in T$
7. $\mathcal{I} : P \times T \to \mathbb{N}^{\infty}$ *is an inhibitor arc weight function*
8. $G : T \to \mathcal{G}$ *is a function from a transition to a guard expression*
9. M_0 *is the initial marking.*



(a) A Colored Petri Net



(b) Petri Net from unfolding Figure 1a

Fig. 1: A Colored Petri Net and its unfolded Petri Net

Let $\mathcal{M}(\mathcal{N})$ be the infinite set of all valid markings for $\mathcal{N}$, where for a marking to be valid we require that $M(p) \in \mathcal{CB}(C(p))$ for all $p \in P$.

When drawing a CPN we represent places as circles, transitions as rectangles and arc expressions as an arrow between the corresponding place and transition, with inhibitor arcs having a circle instead of an arrowhead. Each place has its color type and multiset according to the initial marking, unless otherwise specified, written next to it, as well as an amount of dots representing the amount of tokens in the place. An example CPN is shown in Figure 1a.

We allow integer ranges as color types s.t. if we have $bct \in BCT$ and $colors(bct) = [n, m]$ then $i \in colors(bct)$ for all $n \leq i \leq m$ where $n, m, i \in \mathbb{N}^0$.

Let $en : \mathcal{M}(\mathcal{N}) \times T \times \mathbb{B} \to \{true, false\}$ be a function that returns whether a transition is enabled in a marking with a certain binding, formally defined as $en(M, t, b) = [\![G(t)]\!]_b \wedge \bigwedge_{p \in P}([\![\mathcal{A}(p, t)]\!]_b \subseteq M(p) \wedge |M(p)| < \mathcal{I}(p, t))$. Let

$en : \mathcal{M}(\mathcal{N}) \times T \to \{true, false\}$ be a function that returns whether a transition is enabled in a marking with some binding s.t. $en(M,t) = \bigvee_{b \in \mathbb{B}} en(M,t,b)$.

Firing an enabled transition t in marking M with binding b leads to the marking M' s.t. $M'(p) = (M(p) \setminus [\![\mathcal{A}(p,t)]\!]_b) \uplus [\![\mathcal{A}(t,p)]\!]_b$ for all $p \in P$. We use $M \xrightarrow{t,b} M'$ to denote firing transition t in marking M with some binding b, $M \xrightarrow{w} M'$ to denote firing multiple transitions sequentially where $w \in (T \times \mathbb{B})^*$, and $M \to^* M'$ if $M \xrightarrow{w} M'$ for some $w \in (T \times \mathbb{B})^*$. Let $\mathcal{R}(\mathcal{N})$ be the set of all reachable markings s.t. $M \in \mathcal{R}(\mathcal{N})$ iff $M_0 \to^* M$.

Let ${}^\bullet t$ defined as ${}^\bullet t = \{p \in P \mid \mathcal{A}(p,t) \neq nil\}$ be the pre-places for t and $t^\bullet$ defined as $t^\bullet = \{p \in P \mid \mathcal{A}(t,p) \neq nil\}$ be the post-places for t.

Let $variables : T \to 2^{\mathbb{V}}$ be a function that returns every variable relevant for the firing of a transition t, defined as:

$$variables(t) = variables(G(t)) \cup \bigcup_{p \in t^\bullet} variables(\mathcal{A}(t,p)) \cup \bigcup_{p \in {}^\bullet t} variables(\mathcal{A}(p,t)).$$

Lemma 1. *Let $b, b' \in \mathbb{B}$ be bindings and $t \in T$ be a transition. If $b(v) = b'(v)$ for all $v \in variables(t)$ then $en(M,t,b) = en(M,t,b')$ and if $M \xrightarrow{t,b} M'$ then $M \xrightarrow{t,b'} M'$.*

The correctness of this is evident, because when evaluating guards or arcs by definition only the mapping of variables that appear in the expression influence the result.

2.2 Reachability and LTL

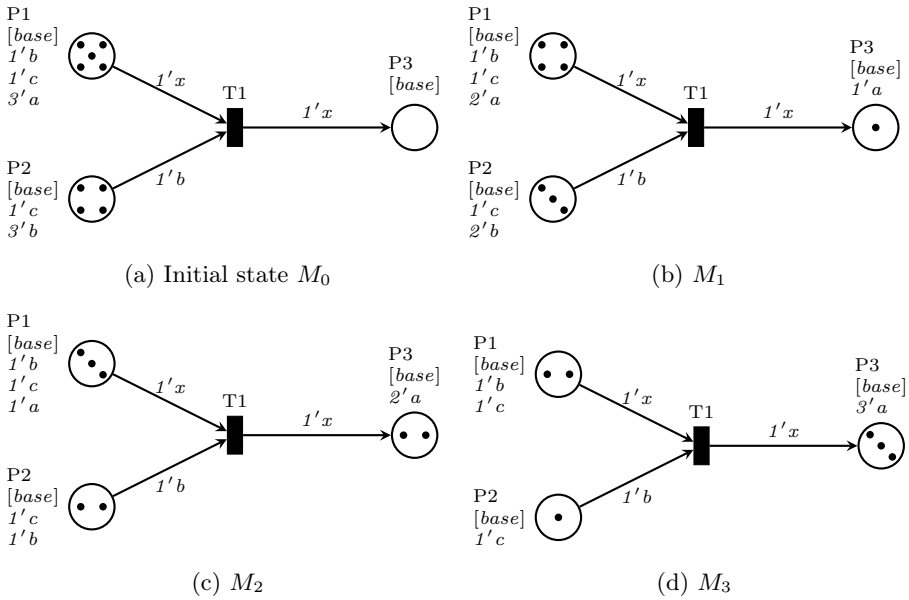
Reachability analysis for CPNs determines whether some marking is reachable from the initial state while LTL checks properties of an infinite sequence of markings. We define the syntax of a query as:

$$\begin{aligned}
\hat{\varphi} &::= \mathbf{A}\ \varphi \mid \mathbf{E}\ \varphi \\
\varphi &::= \gamma \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathbf{F}\ \varphi \mid \mathbf{G}\ \varphi \mid \mathbf{X}\ \varphi \mid \varphi\ \mathbf{U}\ \varphi \\
\gamma &::= deadlock \mid \gamma^c \mid \gamma^f \\
\gamma^c &::= p \bowtie n \mid true \mid false \mid \gamma^c \wedge \gamma^c \mid \gamma^c \vee \gamma^c \mid \neg\gamma^c \\
\gamma^f &::= t \mid true \mid false \mid \gamma^f \wedge \gamma^f \mid \gamma^f \vee \gamma^f \mid \neg\gamma^f
\end{aligned}$$

$$\text{where } t \in T,\ p \in P,\ n \in \mathbb{N} \text{ and } \bowtie \in \{<, \leq, \geq, >, =, \neq\}$$

Here φ represents an LTL query and γ represents a proposition, where γ^c is a cardinality proposition and γ^f is a fireability proposition. A query containing only one type of propositions is a fireability- or cardinality-query. Let Γ denote the set of all propositions.

When evaluating γ on marking $M \in \mathcal{M}(\mathcal{N})$ we replace p with $|M(p)|$, t with $en(M,t)$, and $deadlock$ with $\bigwedge_{t \in T} \neg en(M,t)$. If γ evaluated on a marking M is $true$ then M satisfies γ, written as $M \models \gamma$.



(a) Initial state M_0 (b) M_1

(c) M_2 (d) M_3

Fig. 2: The run $\pi_{ex} = M_0M_1M_2M_3M_3\ldots$ corresponding to $M_0 \xrightarrow{T1,[x\mapsto a]} M_1 \xrightarrow{T1,[x\mapsto a]} M_2 \xrightarrow{T1,[x\mapsto a]} M_3$

LTL is evaluated on an infinite sequence of markings called a *run*. Let $\pi = M_0M_1M_2\ldots$ be a *run* beginning from the initial marking. For every marking M_i either there exists $M_i \xrightarrow{t\in T, b\in\mathbb{B}} M_{i+1}$ or $M_i \models \mathit{deadlock}$ in which case $M_i = M_{i+1}$.

Let π^n refer to the sequence beginning at the nth element s.t. $\pi^2 = M_2M_3\ldots$. The rules for determining whether $\pi \models \varphi$ are:

$$
\begin{aligned}
\pi \models \gamma &\text{ iff } M_0 \models \gamma \\
\pi \models \varphi_1 \wedge \varphi_2 &\text{ iff } \pi \models \varphi_1 \wedge \pi \models \varphi_2 \\
\pi \models \varphi_1 \vee \varphi_2 &\text{ iff } \pi \models \varphi_1 \vee \pi \models \varphi_2 \\
\pi \models \neg\varphi &\text{ iff } \pi \not\models \varphi \\
\pi \models \mathbf{F}\ \varphi &\text{ iff } \pi^i \models \varphi \text{ for some } i \geq 0 \\
\pi \models \mathbf{G}\ \varphi &\text{ iff } \pi^i \models \varphi \text{ for all } i \geq 0 \\
\pi \models \mathbf{X}\ \varphi &\text{ iff } \pi^1 \models \varphi \\
\pi \models \varphi_1\ \mathbf{U}\ \varphi_2 &\text{ iff } \exists j \in \mathbb{N}^0.\ \pi^j \models \varphi_2 \text{ s.t. } \forall i \in \mathbb{N}^0, i < j.\ \pi^i \models \varphi_1
\end{aligned}
$$

Example 7. Let $\mathcal{N}_{ex}$ and π_{ex} be the CPN and run seen in Figure 2 and let $\varphi_{ex} = \mathbf{F}\ P1 = 4 \wedge \mathbf{G}\ P2 \geq 1$. We have $M_1 \models P1 = 4$ and $M \models P2 \geq 1$ for every marking M in π_{ex}, this combined means $\pi_{ex} \models \varphi_{ex}$.

LTL is only defined on a single run, but CPNs allow many different runs depending on which transitions are fired. We disambiguate this in our query

syntax with $\hat{\varphi}$ where every query has a quantifier. For the existential quantifier **E**, a query is satisfied if there exists a single run where φ is satisfied, while for the universal quantifier **A** the query is satisfied if every possible run satisfies φ.

A reachability query can be defined as a subset of LTL where we only allow queries in the form of **E F** γ or **A G** γ. Let Φ denote the set of all reachability queries, Φ_f the set of all reachability-fireability queries, and Φ_c the set of all reachability-cardinality queries.

It is not necessary for our algorithms to be able to solve every kind of proposition, as conjunction and disjunction can be rewritten according to De Morgan's laws and universal- and existential-queries can be rewritten as the other, notably we have **AG** $\gamma \equiv \neg$**EF** $\neg\gamma$.

Example 8. Let $\mathcal{N}_{ex}$ be the CPN in Figure 1a then $\mathcal{N}_{ex} \not\models$ **EF** $\mathrm{P1} = 3 \wedge \mathrm{P3} < 2$, $\mathcal{N}_{ex} \models$ **EF** $\mathrm{T1} \wedge \mathrm{P2} \neq 0$ and $\mathcal{N}_{ex} \not\models$ **AG** $\neg$*deadlock*.

2.3 Nondeterministic Büchi Automaton

We use a *nondeterministic Büchi automaton* (NBA) [2] when evaluating LTL queries, using well-known results adapted to the semantics of our CPNs. Recall that Γ is the set of all γ propositions.

Definition 2. *A nondeterministic Büchi automaton is defined as a tuple* $\mathscr{B} = (Q, \delta, Q_0, F)$ *where*

1. Q *is a set of states,*
2. $\delta \subseteq Q \times \Gamma \times Q$ *is set of transitions s.t. for all* $q \in Q$ *there exists only a finite amount of* $\gamma \in \Gamma$ *and* $q' \in Q$ *where* $(q, \gamma, q') \in \delta$*,*
3. $Q_0 \subseteq Q$ *is a set of initial states, and*
4. $F \subseteq Q$ *is a set of accepting states.*

If there exists a transition $(q, \gamma, q') \in \delta$ we write $q \xrightarrow{\gamma} q'$ or $q \rightarrow q'$. An NBA accepts an infinite sequence of proposition sets $A_0A_1A_2 \cdots \in (2^\Gamma)^\omega$ iff there exists an infinite sequence of states $q_0q_1q_2 \ldots$ s.t.

- $q_0 \in Q_0$,
- $q_i \in F$ for an infinite amount of $i \in \mathbb{N}^0$, and
- $q_i \xrightarrow{\gamma} q_{i+1}$ where $\gamma \in A_i$ for all $i \in \mathbb{N}^0$.

Theorem 1 ([2]). *Let* φ *be an LTL query. There exists an NBA* $\mathscr{B}_\varphi$ *with a finite amount of states s.t. if we have an infinite run* $\pi = M_0M_1M_2 \ldots$ *and an infinite sequence of proposition sets* $w = A_0A_1A_2 \ldots$ *where* $A_i = \{\gamma \in \Gamma \mid M_i \models \gamma\}$ *then* $\mathscr{B}_\varphi$ *accepts* w *iff* $\pi \models \varphi$.

We can create a product automaton [2] which combines an NBA and a CPN into a single NBA.

Definition 3. *Let* $\mathcal{N} = (P, T, CT, \mathbb{V}, C, \mathcal{A}, \mathcal{I}, G, M_0)$ *be a CPN and let* $\mathscr{B} = (Q, \delta, Q_0, F)$ *be an NBA. We define a product automaton* $\mathcal{N} \otimes \mathscr{B} = (Q^p, \delta^p, Q_0^p, F^p)$ *where*

1. $Q^p = \mathcal{M}(\mathcal{N}) \times Q$
2. $\delta^p = \{((M,q),\gamma,(M',q')) \mid t \in T, b \in \mathbb{B}.\ M \xrightarrow{t,b} M' \wedge M \models \gamma \wedge q \xrightarrow{\gamma} q'\} \cup \{((M,q),\gamma,(M,q')) \mid q \xrightarrow{\gamma} q' \wedge M \models \mathit{deadlock} \wedge M \models \gamma\}$
3. $Q_0^p = \{M_0\} \times Q_0$
4. $F^p = \mathcal{M}(\mathcal{N}) \times F$.

The state of the NBA is synchronized with the CPN and a transition can only be taken if it is valid in both the NBA and the CPN. This leads us to the key theorem that allows us to use an NBA to verify LTL queries.

Theorem 2 ([2]). *Let $\mathcal{N}$ be a CPN, φ an LTL query, and $\mathscr{B}^p = \mathcal{N} \otimes \mathscr{B}_{\neg\varphi}$ the corresponding product automaton. There exists a sequence accepted by $\mathscr{B}^p$ iff $\mathcal{N} \not\models \boldsymbol{A}\ \varphi$.*

2.4 Petri Net

We see a CPN $\mathcal{N}^{PT}$ as a PN if $\mathcal{N}^{PT} = (P, T, CT, \mathbb{V}, C, \mathcal{A}, \mathcal{I}, G, M_0)$, where $CT = \{\mathbf{dot}\}$, $color(\mathbf{dot}) = \{\bullet\}$, $\mathbb{V} = \emptyset$ and $[\![\mathcal{G}(t)]\!]_\perp = true$ for all $t \in T$. A PN can therefore be represented by a 5-tuple $\mathcal{N}^{PT} = (P, T, \mathcal{A}, \mathcal{I}, M_0)$.

An example of a PN can be seen in Figure 1b, since there is only one color type, the places are not annotated with multisets.

2.5 Unfolding

When answering queries for CPNs the classical method is unfolding [1]. We will describe the unfolding method documented in [16], but multiple optimisations to unfolding can and have been made in the unfolding engine that we will use in our experiments, some of which are explained in [3].

Definition 4. *Let CPN $\mathcal{N} = (P, T, CT, \mathbb{V}, C, \mathcal{A}, \mathcal{I}, G, M_0)$, then the unfolded PN is $\mathcal{N}^u = (P^u, T^u, \mathcal{A}^u, \mathcal{I}^u, M_0^u)$ where:*

1. $P^u = \{p_{cs} \mid p \in P \wedge cs \in colors(C(p))\} \cup \{p_{sum} \mid p \in P\}$
2. $T^u = \{t_b \mid b \in \mathbb{B}, t \in T$ *where* $[\![G(t)]\!]_b = true\}$
3. $\mathcal{A}^u(t_b, p_{cs}) = [\![\mathcal{A}(t,p)]\!]_b(cs)'\bullet$ *and* $\mathcal{A}^u(p_{cs}, t_b) = [\![\mathcal{A}(p,t)]\!]_b(cs)'\bullet$ *for all $p_{cs} \in P^u$ and $t_b \in T^u$.*
4. *For sum places we have* $\mathcal{A}^u(p_{sum}, t_b) = |[\![\mathcal{A}(p,t)]\!]_b|'\bullet$ *and* $\mathcal{A}^u(t_b, p_{sum}) = |[\![\mathcal{A}(t,p)]\!]_b|'\bullet$ *for all $p_{sum} \in P^u$ and $t_b \in T^u$*
5. $\mathcal{I}^u(p_{sum}, t_b) = \mathcal{I}(p,t)$ *for all $p_{sum} \in P^u$ and $t_b \in T^u$*
6. $M_0^u(p_{cs}) = M_0(p)(cs)'\bullet$ *for all $p_{cs} \in P^u$ and* $M_0^u(p_{sum}) = |M_0(p)|'\bullet$

This results in a PN where for each place in the CPN a place is created for every valid color-sequence and a sum place used for inhibitor arcs where p_{sum} has an amount of tokens equal to the sum of all tokens in p of any color. A new transition is made for every transition and each binding that satisfies its guard. Arc expressions are transformed to multisets of $\bullet$ depending on the color-sequence of the place.

When unfolding the example CPN in Figure 1a we get the PN in Figure 1b. Transitions that only have in- and out-going arcs that evaluate to the empty multiset are not shown. Although the example CPN is small the unfolded net is already graphically more unclear and significantly larger, which showcases the graphical advantage to using CPNs.

We can also unfold queries, if we have a query $\hat{\varphi}$ then $\hat{\varphi}^u$ has any $p \in P$ replaced by p_{sum} and $t \in T$ replaced by $\bigvee_{b \in \mathbb{B}} t_b$.

Example 9. Let $\mathcal{N}_{ex}^u$ be the PN in Figure 1b. Reusing Example 8, let the queries $\phi_1 \equiv \mathbf{EF}\ \mathrm{P1} = 3 \wedge \mathrm{P3} < 2$ and $\phi_2 \equiv \mathbf{EF}\ \mathrm{T1} \wedge \mathrm{P1} \neq 0$. Then we have $\phi_1^u \equiv \mathbf{EF}\ \mathrm{P1_{sum}} = 3 \wedge \mathrm{P3_{sum}} < 2$ and $\phi_2^u \equiv \mathbf{EF}\ (\mathrm{T1}_{[\mathrm{x}\mapsto\mathrm{a}]} \vee \mathrm{T1}_{[\mathrm{x}\mapsto\mathrm{b}]} \vee \mathrm{T1}_{[\mathrm{x}\mapsto\mathrm{c}]}) \wedge \mathrm{P1_{sum}} \neq 0$ s.t. $\mathcal{N}_{ex}^u \not\models \phi_1^u$ and $\mathcal{N}_{ex}^u \models \phi_2^u$.

Based on how we have defined unfolding, there is equivalence between colored and unfolded nets and queries as stated by the following theorem.

Theorem 3 ([3]). *Let $\mathcal{N}$ be a CPN, $\hat{\varphi}$ a query, $\mathcal{N}^u$ the unfolded PN and $\hat{\varphi}^u$ the unfolded query, then $\mathcal{N}^u \models \hat{\varphi}^u$ iff $\mathcal{N} \models \hat{\varphi}$.*

Theorem 1 in [3] states that the CPN $\mathcal{N} = (P, T, CT, \mathbb{V}, C, \mathcal{A}, \mathcal{I}, G, M_0)$ and the unfolded net $\mathcal{N}^u = (P^u, T^u, \mathcal{A}^u, \mathcal{I}^u, M_0^u)$ are bisimilar given the labelling function $\ell(t_b) = t$ for all $t_b \in T^u$. This combined with how the query is unfolded naturally leads to the theorem.

3 Improving the successor generator

When exploring the statespace for CPNs, a successor generator is used to find valid successor markings for a marking $M \in \mathcal{M}$. This can be done naively by iterating through every binding $B \in \mathbb{B}$ and every transition $t \in T$. A naive successor generator can be seen in Algorithm 1. Since CPNs can have many possible bindings per transition, to avoid unnecessary computation, our successor generator returns a single successor when given a marking. Therefore, to ensure that the successor generator will return the successor for each valid transition-binding pair exactly once, we keep track of the bindings and transitions already used by the successor generator for the given marking, in a global map `markingState`. In the pseudocode we assume that sets have some ordering defined by the function *next*, which given an element and a set returns the next element in the ordering or $\top$ if given the last element. We also have *first* which given a set returns the first element in the ordering.

Generating and evaluating unnecessary bindings is computationally expensive, so it is desirable to have techniques making the successor generator iterate through as few unnecessary bindings as possible.

3.1 Color constraint approximation

We propose and implement an approach that avoids generating unnecessary bindings by using a simple heuristic to over-approximate the amount of valid

Algorithm 1: A naive successor generator

```
1  markingState; /* markingState is a globally available map. It maps a
     marking to a transition-binding pair, with initial value
     (first(T), first(𝔹)). */
2  𝓜 ∪ {⊤} NaiveSuccessorGenerator (M)
       Input  : A marking M ∈ 𝓜
       Output : The next reachable marking from M, if none exists then ⊤
3      (t, b) ← markingState[M];
4      while t ≠ ⊤ do
5          while b ≠ ⊤ do
6              b ← next(b, 𝔹);
7              if M -t,b-> M' for some M' then
8                  b ← next(b, 𝔹));
9                  markingState[M] ← (t, b);
10                 return M';
11             end
12         end
13         (t, b) ← (next(t, T), first(B);
14     end
15     markingState[M] ← (t, b);
16     return ⊤;
```

bindings. In this section, we will use a fixed transition t and marking M s.t. a binding b is valid if $en(M, t, b) = true$. The goal is to check each arc expression and the tokens in the relevant places to avoid generating bindings where a variable is mapped to a color s.t. the binding cannot be valid. We do this by creating a function $restrict : \mathbb{V} \to 2^{\mathbb{C}}$ that given a variable returns a restricted set of colors for v s.t. $restrict(v) \subseteq colors(type(v))$ and if there exists $M \xrightarrow{t,b} M'$ then $b(v) \in restrict(v)$ for all $v \in variables(t)$, with the aim of $restrict(v)$ having the lowest cardinality possible.

Example 10. Using the CPN in Figure 3 for t to be enabled, the arc expression between *P1* and t requires a binding b s.t. $[\![1'(x{+}{+}, y)]\!]_b$ is a subset of $M(P1)$. This is only possible if $b(x) \in \{1, 3\}$ and $b(y) \in \{3, 5\}$. Therefore, we want to have $restrict(x) = \{1, 3\}$ and $restrict(y) = \{3, 5\}$.

We define a constraint for a variable $v \in \mathbb{V}$ as a pair (i, o) where $i \in \mathbb{N}^0$ and $o \in \mathbb{Z}$. The first element, i, is the position of the color in the color sequence that is being constrained and the second element, o, is the difference between the value of the variable and the evaluated value. The set of all possible constraints is denoted ϑ, we include a special constraint $\top$ signifying that the variable is unconstrained. We use $\top$ when a relevant variable is present in a subtraction expression.

Let *offset* be a function that given an expression β_i returns the combined value of every increment/decrement, s.t. using the CPN from Figure 3 we have $offset(x{+}{+}) = 1$, $offset(y) = 0$, and e.g. $offset(x{+}{+}{-}{-}{-}{-}) = -1$.

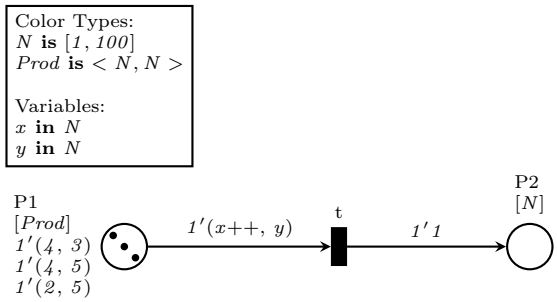


Fig. 3: CPN with clear constraints

Let $AEC : \mathbb{V} \times AE \to 2^{\vartheta}$ be a function that given a variable and an arc expression gathers all the constraints that the arc expression imposes on the variable. We recursively define AEC as:

$$
\begin{aligned}
AEC(v, nil) &= \emptyset \\
AEC(v, m'\alpha) &= \{(i, \mathit{offset}(\beta_i)) \mid i \in \mathbb{N}, i \leq n, v \in \mathit{variables}(\beta_i)\} \\
&\quad \text{where } \alpha = \beta_1 ... \beta_n \\
AEC(v, m * ae) &= AEC(v, ae) \\
AEC(v, ae_1 + ae_2) &= AEC(v, ae_1) \cup AEC(v, ae_2) \\
AEC(v, ae_1 - ae_2) &= \begin{cases} \{\top\} & \text{if } v \in \mathit{variables}(ae_1 - ae_2) \\ \emptyset & \text{otherwise} \end{cases}
\end{aligned}
$$

Crucially for runtime efficiency AEC does not depend on a marking, so the constraints only need to be computed once.

Lemma 2. *Let $p \in P$ be a place, $ae = \mathcal{A}(p, t)$ an arc expression, $v \in \mathit{variables}(ae)$ a variable, and $b \in \mathbb{B}$ a binding. If $[\![ae]\!]_b \subseteq M(p)$ then either $\top \in AEC(v, ae)$ or for some constraint $(i, o) \in AEC(v, ae)$ there exists a multiset element $n'(c_1, ..., c_m) \in M(p)$ s.t. $c_i = b(v) + o$.*

Proof. We prove by structural induction on ae. Our inductive hypothesis states: if $[\![ae]\!]_b \subseteq M(p)$ then for every $v \in \mathit{variables}(ae)$ either $\top \in AEC(v, ae)$ or for some constraint $(i, o) \in AEC(v, ae)$ there exists a multiset element $n'(c_1, ..., c_m) \in M(p)$ s.t. $c_i = b(v) + o$.
For the base cases we have:

- $ae = nil$, which is trivially correct as $\mathit{variables}(nil) = \emptyset$.
- $ae = m'\alpha$, where $\alpha = \beta_1 ... \beta_n$. If we have $v \in \mathit{variables}(ae)$ then there must be an $i \in \mathbb{N}$ s.t. $v \in \mathit{variables}(\beta_i)$ and $(i, \mathit{offset}(\beta_i)) \in AEC(v, m'\alpha)$. When evaluating an arc expression with binding b, a sub-expression with syntax $\beta ::= v(++ \mid --)^*$ clearly evaluates to $b(v) + \mathit{offset}(\beta)$, therefore since $m'\alpha$ evaluates to a multiset with one element if $[\![m'\alpha]\!]_b \subseteq M(p)$ then there must exist an element $n'(c_1, ..., c_i) \in M(p)$ s.t. $c_i = b(v) + \mathit{offset}(\beta_i)$ and the case must be correct.

For the inductive cases we have:

- $ae = m * ae'$, we know the factor m only affects the cardinality of the multiset when evaluated. Since cardinality is non-negative we know that if $[\![m * ae']\!]_b \subseteq M(p)$ then $[\![k * ae']\!]_b \subseteq M(p)$ for all $k \leq m$ where $m, k \in \mathbb{N}$. According to our inductive hypothesis we have that $AEC(v, ae')$ is correct, so $AEC(v, m * ae')$ must also be correct.
- $ae = ae_1 + ae_2$, we know that both ae_1 and ae_2 evaluate to multiset, which because multiset only have non-negative cardinalities means if $[\![ae_1 + ae_2]\!]_b \subseteq M(p)$ then $[\![ae_1]\!]_b, [\![ae_2]\!]_b \subseteq M(p)$ and if $[\![ae_1]\!]_b \not\subseteq M(p)$ or $[\![ae_2]\!]_b \not\subseteq M(p)$ then $[\![ae_1 + ae_2]\!]_b \not\subseteq M(p)$. Following from that and our inductive hypothesis returning $AEC(v, ae_1) \cup AEC(v, ae_2)$ must be correct.
- $ae = ae_1 - ae_2$, where the first subcase is trivially correct as it does not restrict bindings and the second subcase is trivially true as $v \notin \mathit{variables}(ae_1 - ae_2)$.

Since the inductive hypothesis holds for all cases the lemma must be correct. □

Let *restrict-place* : $\mathbb{V} \times 2^{\vartheta} \times P \to 2^{\mathbb{C}}$ be a function that takes a variable, a set of constraints, and a place and returns an over-approximated set of possible colors for the variable according to those constraints, it is defined as:

$$
\begin{aligned}
\mathit{restrict\text{-}place}(v, \emptyset, p) &= \emptyset \\
\mathit{restrict\text{-}place}(v, \{\top\}, p) &= \mathit{colors}(\mathit{type}(v)) \\
\mathit{restrict\text{-}place}(v, \{(i, o)\}, p) &= \{c_i - o \mid n'(c_1, \ldots, c_i, \ldots) \in M(p)\} \\
\mathit{restrict\text{-}place}(v, \{\theta_1, \theta_2, ..., \theta_m\}, p) &= \mathit{restrict\text{-}place}(v, \{\theta_1\}, p) \cup \\
&\quad \mathit{restrict\text{-}place}(v, \{\theta_2, ..., \theta_m\}, p)
\end{aligned}
$$

Lemma 3. *Let $p \in P$ be a place, $v \in \mathit{variables}(\mathcal{A}(p, t))$ a variable, and $b \in \mathbb{B}$ a binding. If $en(M, t, b) = \mathit{true}$ then $b(v) \in \mathit{restrict\text{-}place}(v, AEC(v, \mathcal{A}(p, t)), p)$.*

Proof. We prove by induction on the cardinality of $AEC(v, \mathcal{A}(p, t))$. Our inductive hypothesis states: if $en(M, t, b) = \mathit{true}$ then $b(v) \in \mathit{restrict\text{-}place}(v, AEC(v, \mathcal{A}(p, t)), p)$.
For the base cases we have:

- $|AEC(v, \mathcal{A}(p, t))| = 0$, which is correct based on Lemma 2 as it follows that t cannot be enabled if $AEC(v, \mathcal{A}(p, t)) = \emptyset$.
- $|AEC(v, \mathcal{A}(p, t))| = 1$, which has two different possibilities:
  (i) $AEC(v, \mathcal{A}(p, t)) = \{\top\}$, which is trivially correct following from the definition of bindings.
  (ii) $AEC(v, \mathcal{A}(p, t)) = \{(i, o)\}$ where $\mathit{restrict\text{-}place}(v, \{(i, o)\}, p)$ returns a set of colors s.t. if $c_i \in \mathit{restrict\text{-}place}(v, \{(i, o)\}, p)$ then there exists a $n'(c_1, ..., c_i, ...) \in M(p)$ where $c_i = b(v) - o$. Notice that this is one of the possibilities in Lemma 2 and the other is trivially not satisfied s.t. if $b(v) \notin \mathit{restrict\text{-}place}(v, \{i, o\}, p)$ then $M(P) \not\subseteq [\![\mathcal{A}(p, t)]\!]_b$ implying $en(M, t, b) = \mathit{false}$.

For the inductive case we have:

- $|AEC(v, \mathcal{A}(p,t))| = m$, s.t. $AEC(v, \mathcal{A}(p,t)) = \{(\theta_1, \theta_2, ..., \theta_m)\}$, where the result of *restrict-place*$(v, AEC(v, \mathcal{A}(p,t)), p)$ consists of two elements, firstly the recursive result of the first element, which we know is correct since it is one of the base cases and the other element is the recursive result of the remaining set which is correct according to our inductive hypothesis. The returned value is the union of those elements, which clearly is correct since union is a monotonic function.

Thus the inductive hypothesis holds. □

Let $restrict : \mathbb{V} \to 2^{\mathbb{C}}$, be a function that takes a variable and returns an over-approximated set of possible colors. It is formally defined as:

$$restrict(v) = \begin{cases} \emptyset & v \notin variables(t) \\ colors(type(v)) & v \notin \bigcup\limits_{p \in {}^\bullet t} variables(\mathcal{A}(p,t)) \\ \bigcap\limits_{p \in {}^\bullet t} restrict\text{-}place(v, AEC(v, \mathcal{A}(p,t)), p) & \text{otherwise} \end{cases}$$

The second case is for variables that only appear in output arcs.

Theorem 4. *Let $b \in \mathbb{B}$ be a binding and $v \in variables(t)$ a variable, if $M \xrightarrow{t,b} M'$ then $b(v) \in restrict(v)$.*

Proof. We have three different cases for $restrict(v)$, the first case is trivially correct as it has $v \notin variables(t)$ as a predicate.

The second case returns $colors(type(v))$, which is trivially correct following from the definition of bindings.

In the third case, by contradiction assume there is a binding b where $b(v) \notin restrict(v)$ and $M \xrightarrow{t,b} M'$, this must mean that for some $p \in {}^\bullet t$ we have $b(v) \notin restrict\text{-}place(v, AEC(v, \mathcal{A}(p,t)), p))$ which cannot be true as according to Lemma 3 that would mean $en(M, t, b) = false$. □

Let $\bar{\mathbb{B}}(M,t)$ be a set of over-approximated bindings for M and t, that can be constructed from the color constraints s.t. if $en(M,t,b) = true$ then $b \in \bar{\mathbb{B}}(M,t)$. If there exists a $v \in variables(t)$ where $restrict(v) = \emptyset$ then $\bar{\mathbb{B}}(M,t) = \emptyset$. Otherwise $\bar{\mathbb{B}}(M,t)$ is a maximal set of bindings, where a binding exists for all permutations of mappings for variables relevant to the transition and every mapping for variables not relevant for the transition is the same for all bindings s.t. for all $b \in \bar{\mathbb{B}}(M,t)$ we have $b(v) \in restrict(v)$ for all $v \in variables(t)$ and $b(v) = first(colors(type(v)))$ for all $v \notin variables(t)$[1]. The correctness of the mappings for variables not relevant for t follows from Lemma 1.

[1] In our implementation bindings only relevant variables are included in a binding.

Algorithm 2: SuccessorGenerator

```
1  markingState; /* markingState is a globally available map. It maps a
     marking M to a transition-binding pair, with initial value
     (first(T),first(B̄(M,t))). */
2  M ∪ {⊤} SuccessorGenerator (M)
       Input  : A marking M ∈ M
       Output : The next reachable marking from M, if none exists then ⊤
3      (t, b) ← markingState[M];
4      while t ≠ ⊤ do
5          if |M(p)| > I(p,t) ∨ |M(p)| < min-cardinality(A(p,t)) for some p ∈ P
           then
6              go to line 16;
7          end
8          while b ≠ ⊤ do
9              b ← next(b, B̄(M,t));
10             if M --t,b--> M' for some M' then
11                 b ← next(b, B̄(M,t)) ;
12                 markingState[M] ← (t, b);
13                 return M';
14             end
15         end
16         (t, b) ← (next(t,T),first(B̄(M,t)));
17     end
18     markingState[M] ← (t, b);
19     return ⊤ ;
```

Example 11. Using the CPN in Figure 3, using our color constraint method, with t and M fixed as $T1$ and M_0, we get:
$AEC(x, 1'(x{+}{+}, y)) = \{(1,1)\}$, $AEC(y, 1'(x{+}{+}, y)) = \{(2,0)\}$,
restrict-place$(x, \{(1,1)\}, P)) =$ *restrict*$(x) = \{1,3\}$,
restrict-place$(y, \{(2,0)\}, P)) =$ *restrict*$(y) = \{3,5\}$, and
$\bar{\mathbb{B}}(M_0, T1) = \{[x \mapsto 1, y \mapsto 3], [x \mapsto 1, y \mapsto 5], [x \mapsto 3, y \mapsto 3], [x \mapsto 3, y \mapsto 5]\}$.
Resulting in four bindings, where naively generating bindings would give 100^2.

3.2 Improved successor generator

Our improved successor generator implementing these optimisations can be seen in Algorithm 2. The optimisations we have implemented have been highlighted in grey. The light gray optimisation is where we only iterate through the constrained possible bindings as explained in Section 3.1. The dark grey optimisation checks inhibitor arcs and whether the amount of tokens in each place can potentially enable the transition before iterating through bindings, which are optimisations from [11].

Theorem 5 ([11]). *Let $M \in \mathcal{M}(\mathcal{N})$ be a marking and let*

$$\begin{aligned} M_1 &= SuccessorGenerator(M) \\ M_2 &= SuccessorGenerator(M) \\ M_3 &= SuccessorGenerator(M) \\ &\vdots \end{aligned}$$

then there is an $i \in \mathbb{N}$ s.t. $M_i = \top$ and $M_j = \top$ for all $j > i$. Let i be the smallest number s.t. $M_i = \top$ then $\{M_1, ...M_{i-1}\} = \{M' \mid t \in T, b \in \mathbb{B}.\ M \xrightarrow{t,b} M'\}$.

Proof (sketch). Notice that whenever a successor is returned, the internal state is saved. Additionally the loop on lines 4-17 iterate through every transition-binding pair with a potentially valid binding and returns a successor if the transition-binding pair is valid. If line 6 is reached then clearly the transition cannot be enabled regardless of binding based on the definition of *min-cardinality* and the definition of inhibitor arcs. □

4 Improvements for reachability verification

Some queries do not require exploring the statespace, as they can be either reduced to a trivial expression or answered with a simplified net. For instance to verify $\hat{\varphi} = \mathbf{AG}$ *true*, our approach in [11] would need to explore the full statespace to verify a trivially *true* query. We want to introduce query simplification to answer some queries without exploring the state space.

4.1 Reachability cardinality simplification

Color over-approximation as currently implemented in TAPAAL [17] is a technique for query simplification avoiding the overhead of unfolding. Our proposed technique is based on the same idea, but increases the cases it can be applied to as well as correcting a bug in the current implementation. This technique only applies to reachability queries, so every query mentioned in this section is a reachability query.

When using over-approximation, we convert the CPN by removing colors and guards resulting in a PN which we call a color ignorant net. Converting a CPN into a color ignorant net is almost trivial except for when arc expressions include subtraction, which needs to be handled carefully to retain all behaviour. When converting a transition with an in- or out-going arc that includes subtraction, we find the minimum and maximum possible cardinality for every arc and assume every value within the range is possible, then for every permutation of input- and output-arcs a new transition is created with corresponding arcs. As we will prove in Lemma 4 the behaviour of the color ignorant net is an over-approximation, where when disregarding colors every marking reachable in the original net is also reachable for the color ignorant net, since the removed elements from the original net such as guards and colors only limit behaviour.

An arc expression ae when evaluated has a cardinality in the interval $[\textit{min-cardinality}(ae), \textit{max-cardinality}(ae)]$. When creating a color ignorant net each transition is replaced by a number of new transitions according to the possible permutations of cardinality for in- and out-going arcs. Let $perm_t : (\{t\} \times P) \cup (P \times \{t\}) \to \mathbb{N}^0$ be a function that takes a place-transition pair and returns the cardinality for the appropriate arc s.t. $\textit{min-cardinality}(\mathcal{A}(t,p)) \leq perm_t(t,p) \leq \textit{max-cardinality}(\mathcal{A}(t,p))$ with an equivalent relation for $perm_t(p,t)$. We denote the set of all valid permutations for a transition $t \in T$ as $Permutations_t$.

Definition 5. *Let CPN $\mathcal{N} = (P, T, CT, \mathbb{V}, C, \mathcal{A}, \mathcal{I}, G, M_0)$, then the color ignorant PN is $\mathcal{N}^i = (P^i, T^i, \mathcal{A}^i, \mathcal{I}^i, M_0^i)$ where:*

1. $P^i = P$
2. $T^i = \{t_{perm_t} \mid t \in T, perm_t \in Permutations_t\}$
3. $\mathcal{A}^i(t_{perm_t}, p) = perm_t(t,p)'\bullet$
4. $\mathcal{A}^i(p, t_{perm_t}) = perm_t(p,t)'\bullet$
5. $\mathcal{I}^i(p, t_{perm_t}) = \mathcal{I}(p,t)$
6. $M_0^i(p) = |M_0(p)|'\bullet$

An example of a CPN and its corresponding color ignorant net can be seen in Figure 4.

Lemma 4. *Given a CPN $\mathcal{N} = (P, T, CT, \mathbb{V}, C, \mathcal{A}, \mathcal{I}, G, M_0)$, and PN $\mathcal{N}^i = (P^i, T^i, \mathcal{A}^i, \mathcal{I}^i, M_0^i)$ then for every $M \in \mathcal{R}(\mathcal{N})$ there exists a $M' \in \mathcal{R}(\mathcal{N}^i)$ where $|M(p)| = |M'(p)|$ for all $p \in P$.*

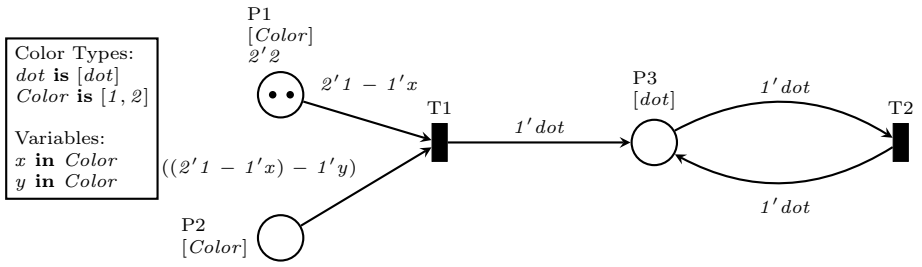
Proof. We prove the lemma by induction on the length of w. Our inductive hypothesis states that for every $M_0 \xrightarrow{w} M$ there exists a $M_0' \xrightarrow{w'} M'$ where $|M(p)| = |M'(p)|$ for all $p \in P$.
For our base case we have:

- $|w| = 0$, where we have $M_0 \xrightarrow{w} M_0$ and $M_0' \xrightarrow{w} M_0'$, which is true by construction.

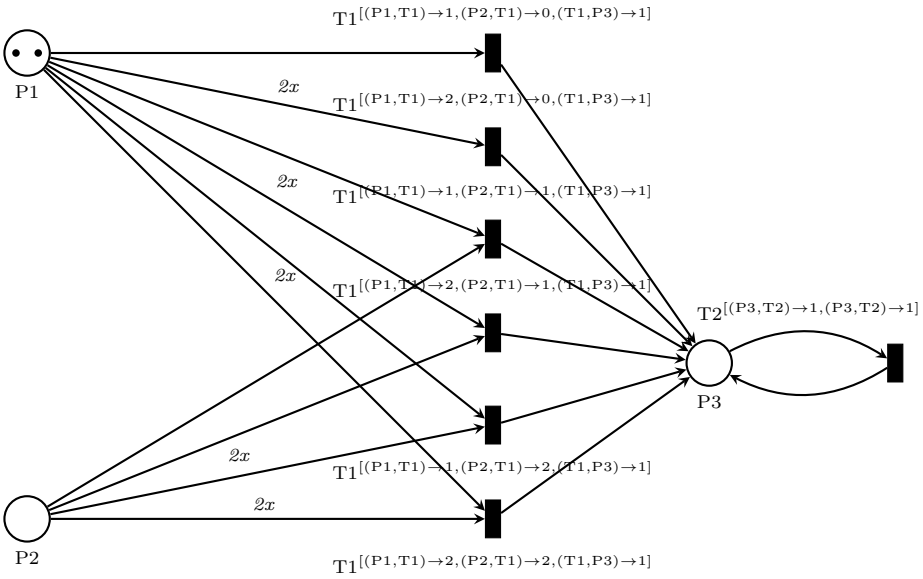
For the inductive case we have:

- $|w| = n$, where $n \in \mathbb{N}$ s.t. $w = (t_1, b_1)...(t_n, b_n)$. Then if $M_0 \xrightarrow{(t_1,b_1)...(t_n,b_n)} M$ we have $M_0 \xrightarrow{(t_1,b_1)...(t_{n-1},b_{n-1})} M_{n-1} \xrightarrow{t_n,b_n} M$. According to the inductive hypothesis there must be a $M'_{n-1} \in \mathcal{R}(\mathcal{N}^i)$ where $|M_{n-1}(p)| = |M'_{n-1}(p)|$ for all $p \in P$. By construction of $\mathcal{N}^i$ we have a $t' \in T^i$ with a trivially *true* guard, $\mathcal{I}(p, t_n) = \mathcal{I}(p, t')$, and $|[\![\mathcal{A}(p, t_n)]\!]_{b_n}| = |[\![\mathcal{A}(p, t')]\!]_\perp| \wedge |[\![\mathcal{A}(t_n, p)]\!]_{b_n}| = |[\![\mathcal{A}(t', p)]\!]_\perp|$ for all $p \in P$ s.t. $M'_{n-1} \xrightarrow{t',\perp} M'$ where $|M(p)| = |M'(p)|$ for all $p \in P$.

Hence the inductive hypothesis holds. □



(a) A Colored Petri Net $\mathcal{N}$



(b) Petri Net $\mathcal{N}^i$ from converting Figure 4a into a color ignorant net

Let $\phi \equiv \mathbf{EF}\ P2 = 4$ and $\phi' \equiv \mathbf{AG}\ \neg T2$.
Then $\mathcal{N}^i \not\models \phi$, $\mathcal{N} \not\models \phi$, $\mathcal{N}^i \not\models \phi'$, and $\mathcal{N} \models \phi'$.

(c) Example queries

Fig. 4: Example CPN, color ignorant net, and queries

When evaluating a query from the original net on a color ignorant net cardinality queries can be directly evaluated since the places are the same. For fireability queries, transitions need to be replaced where any $t \in T$ is replaced with $\bigvee_{perm_t \in Permutations_t} t_{perm_t}$, we can observe that since the color ignorant net has no colors, there must be a transition t^i where for every transition-binding pair it returns the minimal cardinality of that arc s.t. when evaluated t^i and $\bigvee_{perm_t \in Permutations_t} t_{perm_t}$ are equivalent.

Over-approximation can be used for cardinality queries relatively simply by utilising Lemma 4, which because cardinality queries only depend on the cardinality of tokens in each place means that if a marking satisfying a cardinality expression cannot be reached for the color ignorant net, then it will also not be reachable in the original net. This leads to the following theorem for cardinality queries:

Theorem 6. *Let γ^c be a cardinality expression, if $\mathcal{N}^i \not\models \boldsymbol{EF}\ \gamma^c$ then $\mathcal{N} \not\models \boldsymbol{EF}\ \gamma^c$, and if $\mathcal{N}^i \models \boldsymbol{AG}\ \gamma^c$ then $\mathcal{N} \models \boldsymbol{AG}\ \gamma^c$.*

Proof. Lemma 4 states that for every $M \in \mathcal{R}(\mathcal{N})$ there exists a $M^i \in \mathcal{R}(\mathcal{N}^i)$ where $|M(p)| = |M^i(p)|$ for all $p \in P$. By definition $\mathcal{N}^i \not\models \mathbf{EF}\ \gamma^c$ means that $M \not\models \gamma^c$ for every $M \in \mathcal{R}(\mathcal{N}^i)$. Cardinality expressions evaluated in a marking M replaces p with $|M(P)|$, so clearly we have $M' \not\models \gamma^c$ for all $M' \in \mathcal{R}(\mathcal{N})$ s.t. $\mathcal{N} \not\models \mathbf{EF}\ \gamma^c$. Likewise $\mathcal{N}^i \models \mathbf{AG}\ \gamma^c$ means that $M \models \gamma^c$ for every $M \in \mathcal{R}(\mathcal{N}^i)$ and clearly $M \models \gamma^c$ for every $M \in \mathcal{R}(\mathcal{N})$ s.t. $\mathcal{N} \models \mathbf{AG}\ \gamma^c$. □

4.2 Reachability fireability simplification

When using over-approximation to verify fireability queries the color ignorant net can never determine that a transition is enabled, since that requires information about specific colors. Therefore we evaluate the sub-expressions of the query to determine whether the result of that sub-expression, when evaluated on the color ignorant net can be used to evaluate the query for the original net.

We want to have a function that given a fireability query returns the result of the query for the original net, if it can be determined via over-approximation.

Let $\Psi : \Phi_f \to \{\top, ?, \bot\}$, be a function that given a reachability-fireability query returns either whether the original net satisfies the query or that it cannot be determined via the color ignorant net, defined as:

$$\Psi(\mathbf{AG}\ t) = \begin{cases} \bot & \text{if } M \not\models t^i \text{ for all } M \in \mathcal{R}(\mathcal{N}^i) \\ ? & \textit{otherwise} \end{cases}$$

$$\Psi(\mathbf{AG}\ \textit{true}) = \top$$

$$\Psi(\mathbf{AG}\ \textit{false}) = \bot$$

$$\Psi(\mathbf{AG}\ \gamma_1^f \wedge \gamma_2^f) = \begin{cases} \top & \text{if } \Psi(\mathbf{AG}\ \gamma_1^f) = \top \wedge \Psi(\mathbf{AG}\ \gamma_2^f) = \top \\ \bot & \text{if } \Psi(\mathbf{AG}\ \gamma_1^f) = \bot \vee \Psi(\mathbf{AG}\ \gamma_2^f) = \bot \\ ? & \text{otherwise} \end{cases}$$

$$\Psi(\mathbf{AG}\ \neg\gamma^f) = \begin{cases} \top & \text{if } \Psi(\mathbf{AG}\ \gamma^f) = \bot \\ \bot & \text{if } \Psi(\mathbf{AG}\ \gamma^f) = \top \\ ? & \text{if } \Psi(\mathbf{AG}\ \gamma^f) = ? \end{cases}$$

$$\Psi(\neg\mathbf{AG}\ \gamma^f) = \Psi(\mathbf{AG}\ \neg\gamma^f)$$

Notice that since we use over-approximation, the propositions need to hold for all states. This effectively means that EF and AG are equivalent when determining whether the color ignorant net can provide an answer to the query.

Theorem 7. *Let $\mathcal{N}$ be a CPN, $\hat{\varphi}$ a reachability-fireability query, and $\mathcal{N}^i$ a converted PN, if $\Psi(\hat{\varphi}) = \top$ then $\mathcal{N} \models \hat{\varphi}$ and if $\Psi(\hat{\varphi}) = \bot$ then $\mathcal{N} \not\models \hat{\varphi}$ and $\mathcal{N} \models \neg\hat{\varphi}$.*

Proof. We prove the theorem by structural induction on $\hat{\varphi}$. Our inductive hypothesis states: if $\Psi(\hat{\varphi}) = \top$ then $\mathcal{N} \models \hat{\varphi}$ and if $\Psi(\hat{\varphi}) = \bot$ then $\mathcal{N} \not\models \hat{\varphi}$ and $\mathcal{N} \models \neg\hat{\varphi}$.
For the base cases we have:

- $\hat{\varphi} = \mathbf{AG}\ t$, where because of Lemma 4, the definition of *min-cardinality*, and the construction of $\mathcal{N}^i$ we can determine that for any $t \in T$ and $M \in \mathcal{R}(\mathcal{N})$ if $M \models t$ then there exists a $M' \in \mathcal{R}(\mathcal{N}^i)$ where $M' \models t^i$. This means that if $en(t^i, M) = \mathit{false}$ for all $M \in \mathcal{R}(\mathcal{N}^i)$ then $en(t, M) = \mathit{false}$ for all $M \in \mathcal{R}(\mathcal{N})$, so the case must be correct.
- $\hat{\varphi} = \mathit{true}$ is trivially correct.
- $\hat{\varphi} = \mathit{false}$ is trivially correct.

For the inductive cases we have:

- $\hat{\varphi} = \mathbf{AG}\ \gamma_1^f \wedge \gamma_2^f$, which has three subcases:
  (i) We have $\Psi(\mathbf{AG}\ \gamma_1^f) = \top$ and $\Psi(\mathbf{AG}\ \gamma_2^f) = \top$, which according to the inductive hypothesis means $\mathcal{N} \models \mathbf{AG}\ \gamma_1^f$ and $\mathcal{N} \models \mathbf{AG}\ \gamma_2^f$ so clearly $\mathcal{N} \models \mathbf{AG}\ \gamma_1^f \wedge \gamma_2^f$.
  (ii) We have either $\Psi(\mathbf{AG}\ \gamma_1^f) = \bot$ or $\Psi(\mathbf{AG}\ \gamma_2^f) = \bot$. If $\Psi(\mathbf{AG}\ \gamma_1^f) = \bot$ then according to the inductive hypothesis $\mathcal{N} \not\models \mathbf{AG}\ \gamma_1^f$, which means $\mathcal{N} \not\models \mathbf{AG}\ \gamma_1^f \wedge \gamma_2^f$. This applies symmetrically to $\Psi(\mathbf{AG}\ \gamma_2^f) = \bot$
  (iii) Evaluates to ?, which is trivially correct.
- $\hat{\varphi} = \mathbf{AG}\ \neg\gamma^f$, which has three subcases.
  (i) We have $\Psi(\mathbf{AG}\ \gamma^f) = \bot$ and according to our inductive hypothesis $\mathcal{N} \models \mathbf{AG}\ \neg\gamma^f$.
  (ii) We have $\Psi(\mathbf{AG}\ \gamma^f) = \top$, which according to the inductive hypothesis means $\mathcal{N} \models \mathbf{AG}\ \gamma^f$ implying $\mathcal{N} \not\models \neg\mathbf{AG}\ \gamma^f$.
  (iii) Evaluates to ?, which is trivially correct.
- $\hat{\varphi} = \neg\mathbf{AG}\ \gamma^f$. There are three different possibilities depending on the value of $\Psi(\mathbf{AG}\ \neg\gamma^f)$. The first possibility is $\Psi(\mathbf{AG}\ \neg\gamma^f) = \top$, where following from our inductive hypothesis $\mathcal{N} \models \mathbf{AG}\ \neg\gamma^f$ implying $\mathcal{N} \models \neg\mathbf{AG}\ \gamma^f$ and $\Psi(\neg\mathbf{AG}\ \gamma^f) = \Psi(\mathbf{AG}\ \neg\gamma^f) = \top$. If we have $\Psi(\mathbf{AG}\ \neg\gamma^f) = \bot$ then according to the inductive hypothesis $\mathcal{N} \models \mathbf{AG}\ \gamma^f$ and clearly $\mathcal{N} \not\models \neg\mathbf{AG}\ \gamma^f$ and $\Psi(\neg\mathbf{AG}\ \gamma^f) = \Psi(\mathbf{AG}\ \neg\gamma^f) = \bot$. The last possibility where $\Psi(\mathbf{AG}\ \neg\gamma^f) = ?$ is trivially correct.

Hence the inductive hypothesis holds. □

Example 12. Using ϕ' from Figure 4c, we have $\Psi(\mathbf{AG}\ \neg T2) \equiv \Psi(\neg\mathbf{AG}\ T2)$ and $\mathcal{N}^i \not\models \mathbf{AG}\ \neg T2^i$ leading to $\Psi(\neg\mathbf{AG}\ T2) = ?$, which means $\mathcal{N}^i$ cannot simplify ϕ'.

Algorithm 3: Product successor generator

```
1  successorState;   /* successorState is a globally available map. It
     maps a marking and NBA state pair (M × Q) to another marking and
     NBA state pair or ⊥, with initial value ⊥. */
2  Q_q ; /* Q_q is the set of NBA states that can be reached from q s.t.
     q' ∈ Q_q iff q → q' where q, q' ∈ Q. */
3  (M × Q) ∪ ⊤ ProductSuccessorGenerator (M, q)
       Input  : A marking M and NBA state q
       Output : The next successor marking and NBA state pair or ⊤ if none
                exist
4      if successorState[(M, q)] = ⊥ then
5          M' = SuccessorGenerator(M, q);          /* Call to Algorithm 2 */
6          q' = first(Q_q);
7          if M' = ⊤ then
8              successorState[(M, q)] ← (M, q');   /* In case of deadlock */
9          else
10             successorState[(M, q)] ← (M', q');
11         end
12     end
13     (M', q') ← successorState[(M, q)];
14     while M' ≠ ⊤ do
15         if q' = ⊤ then
16             q' ← first(Q_q);
17             M' ← SuccessorGenerator(M, q);   /* Call to Algorithm 2 */
18         end
19         if q --γ--> q' ∧ M ⊨ γ then
20             successorState[(M, q)] ← (M', next(q', Q_q));
21             return (M', q');
22         end
23         q' ← next(q', Q_q);
24     end
25     successorState[(M, q)] ← (M', q);
26     return ⊤;
```

Let $simplify : \Phi \to \{\top, ?, \bot\}$ be a function that takes a reachability query and either returns whether it is satisfied in $\mathcal{N}$ based on over-approximation or ? if it cannot be determined. It uses Theorem 6 for cardinality queries, Theorem 7 for fireability queries, and returns ? for other query types or if the theorem cannot provide an answer. This function is used by our worklist algorithm to determine whether the query can be simplified, as seen in Algorithm 5.

5 LTL verification

When verifying LTL queries we make use of a Büchi automaton, so the state space we need to explore consists of product states containing an NBA state and a marking. Therefore the naive worklist algorithm used for reachability queries cannot be used. Following a standard LTL approach [7], we implement a new

successor generator that generates product state successors and the NDFS search strategy, which explores the product state space finding accepting product states and determines if they are part of an accepting loop.

5.1 Product successor generator

Our implementation of a product successor generator can be seen in Algorithm 3, which iteratively returns a new successor state every time it is called with the same state. If no new successors are available it will return $\top$.

Notice that the call to *SuccessorGenerator* passes both a marking and an NBA state, while in Algorithm 2 the algorithm only takes a marking. To generate all product state successors, we have `markingState` map from product states instead of just markings.

Lemma 5. *Let $M \in \mathcal{M}$ be a marking, $q \in Q$ an NBA state and*

$$\begin{aligned} s_0 &= \mathit{ProductSuccessorGenerator}(M, q) \\ s_1 &= \mathit{ProductSuccessorGenerator}(M, q) \\ s_2 &= \mathit{ProductSuccessorGenerator}(M, q) \\ &\vdots \end{aligned}$$

then there is an $i \in \mathbb{N}$ s.t. $s_i = \top$ and $s_j = \top$ for all $j > i$. Let i be the smallest number s.t. $s_i = \top$ then $\{s_1, ..., s_{i-1}\} = \{(M', q') \in \mathcal{M}(\mathcal{N}) \times Q \mid (M, q) \rightarrow (M', q')\}$.

Proof (sketch). Notice that whenever a product state is returned, the internal state is saved. When called the first time, the control structure on lines 4-12 ensure that successors are generated if there is a deadlock. Additionally the loop on lines 14-24 iterate through every proposition with a valid transition for the NBA and returns every state reachable for the product NBA. Finding the successor markings on line 17 uses the *SuccessorGenerator*, which according to Theorem 5 will return every successor marking. □

5.2 Nested depth-first search

Using the product successor generator the product state space can be explored. To answer queries we need to find a sequence of states with an infinite amount of accepting NBA states. We achieve this by implementing nested depth-first search [6] using two different worklist algorithms. Algorithm 4 takes a marking and NBA state pair and returns whether there exists a loop starting from and ending in the given state, s.t. if the given pair is part of a sequence of accepting states and *true* is returned, an infinite sequence of accepting NBA states exist.

The final worklist algorithm is given a CPN and a query and returns whether the query is satisfied and can be seen in Algorithm 5. To determine whether the query is satisfied, it uses the product successor generator to search the state

Algorithm 4: Nested depth-first search

```
1  passed; /* passed is a globally available set containing triples of
     a marking, NBA state, and a number s.t. passed ⊆ M(N) × Q × {0,1}
     */
2  bool ndfs (M, q)
       Input  : A marking M and an NBA state q
       Output : A boolean representing if the marking and NBA state is part of
                an accepting loop
3      oldMarkingState ← markingState;
4      Set markingState to its initial value;        /* The successor generator
        needs to generate every state and should be restored. */
5      oldSuccessorState ← successorState;
6      Set SuccessorState to its initial value;       /* The product successor
        generator needs to generate every product state and should be
        restored. */
7      targetState ← (M, q);
8      Add (M, q, 1) to passed;
9      Let waiting be a stack with initial element (M, q);
10     while waiting ≠ ∅ do
11         Select (M, q) from waiting;
12         succ ← ProductSuccessorGenerator(M, q);    /* Call to Algorithm 3
            */
13         if succ ≠ ⊤ then
14             (M', q') ← succ;
15             if (M', q', 1) ∉ passed then
16                 Add (M', q', 1) to passed;
17                 Push (M', q') to waiting;
18             else if (M', q') = targetState then
19                 markingState ← oldMarkingState;
20                 successorState ← oldSuccessorState;
21                 return true;
22             end
23         else
24             Pop (M, q) from waiting;
25         end
26     end
27     markingState ← oldMarkingState;
28     successorState ← oldSuccessorState;
29     return false;
```

space for reachable accepting states, each accepting state is then checked for the existence of an accepting loop using Algorithm 4. If a reachable accepting loop is found s.t. the query is satisfied, the algorithm will return *true*. We use our query simplification for reachability queries, which can be seen in gray.

Theorem 8 ([6]). *Let $\mathcal{N}$ be a CPN, $\hat{\varphi}$ an LTL query, and LTLChecker be Algorithm 5, then $\mathcal{N} \models \hat{\varphi}$ iff LTLChecker$(\hat{\varphi}) = true$.*

Algorithm 5: LTL Checker

```
1  passed; /* passed is a globally available set containing triples of
     a marking, NBA state, and a number s.t. passed ⊆ M(N) × Q × {0,1}
     */
2  bool LTLChecker (N, φ̂)
       Input  : A CPN tuple N = (P, T, CT, 𝕍, C, A, I, G, M_0) and a query
                φ̂ = A φ
       Output : A boolean representing N ⊨ φ̂
3      if φ̂ ∈ Φ then
4          answer ← simplify(φ̂);              /* Using over-approximation */
5          if answer ≠ ? then
6              return answer = ⊤;
7          end
8      end
9      (Q, δ, Q_0, F) ← B_{¬φ};
10     passed ← {M_0} × Q_0 × {0};
11     waiting ← {M_0} × Q_0;
12     while waiting ≠ ∅ do
13         Select (M, q) from waiting;
14         succ ← ProductSuccessorGenerator(M, q);   /* Call to Algorithm 3
             */
15         if (succ ≠ ⊤) then
16             (M', q') ← succ;
17             if (M', q', 0) ∉ passed then
18                 if q ∈ F ∧ ndfs(M, q) then       /* Call to Algorithm 4 */
19                     return true;
20                 end
21                 Add {(M', q', 0)} to passed;
22                 Add (M', q') to waiting;
23             end
24         else
25             Remove (M, q) from waiting;
26         end
27     end
28     return false;
```

In practice for reachability queries we use the Reachability Search from [11] with our query simplification added, seen as the grey optimisation in Algorithm 5.

6 Experiments

We implemented our improvements and techniques in TAPAAL written in C++, using the implementation from [11] as a baseline. The source code is available on GitHub [12]. We benchmark our implementation against the current implementation of TAPAAL [8] and our previous implementation [11]. Each configuration is benchmarked on all of the colored models from MCC'24 [18] using the cardinality- and fireability-queries for reachability or LTL. There are 272 models

in the dataset, each model having 16 of each query type, giving a total of 8.704 queries each for LTL and reachability. Each query is assigned a single core of an AMD EPYC 9334 with 16 GiB of available memory and a five minute timeout.

The experiments showed complete consistency of answers between our implementation, TAPAAL, and the accepted answers from MCC'24 [21,20,22,23].

In some experiments we additionally split queries into categories of positive- and negative-results, where a query with a positive result has a run that satisfies the query while negative results do not.

6.1 Implementation

In addition to the techniques in this thesis there have been general implementation improvements that contribute to the increased performance. Some of these improvements include better encoding of the passed list by introducing different ways of encoding the tokens in a place and for each place heuristically using the one that needs the fewest bytes, compiling queries by converting place names into the corresponding array index so a hash table lookup is avoided, and a more efficient representation of product colors by encoding it as an unsigned integer instead of a vector of unsigned integers.

In addition to the code improvements and new techniques, we helped discover and document existing bugs in TAPAAL related to e.g. unfolding, parsing of PNML models, and colored reductions, as well as develop new features for the explicit colored engine.

For reachability queries we implemented the ability to reconstruct traces when a positive result is found. Instead of storing traces for each marking, we create a structure akin to a linked list that stores which transition and binding was taken to reach each marking. The binding is stored in a compressed format, so trace reconstruction only has a few extra bytes overhead per reachable marking. This makes it possible for the TAPAAL GUI to visualise a trace answering a CPN query.

We also implemented a simulation mode with a communication protocol, allowing the TAPAAL GUI to call our engine, supply a CPN and marking and receive a list of valid transition-binding pairs of which it can fire one and get the successor state. In the future, the TAPAAL GUI can use this to visualise a CPN and allow the user to explore the state space themselves.

6.2 Reachability experimentation

In our previous implementation there was substantial difference in the amount of solved reachability queries depending on whether the result was positive or negative. To identify potential solutions and the most promising techniques, we compare TAPAAL's different techniques individually with the results in Table 1, we have also included the previous- and current-implementation of our approach for comparison.

Name	Answers	Cardinality		Fireability		Naive unique
		+	-	+	-	
Explicit full*	7287	1425	2622	2626	614	3153
Full	6691	1192	2553	2356	590	2607
Query simplification [4]	6220	1122	2416	2208	474	2064
Initial marking simplification [4]	4779	1127	1117	2208	327	621
Explicit old* [11]	4682	1383	554	2510	235	775
Structural reduction [5]	4553	1151	827	2250	325	400
Partial order reduction [5]	4526	1127	793	2229	377	416
Colored structural reduction [9]	4423	1138	737	2238	310	266
Color partitioning [3]	4369	1112	705	2190	362	256
Color fix-point [9]	4297	1109	602	2301	285	133
Variable symmetry [9]	4199	1114	586	2218	281	23
Interval [9]	4195	1114	586	2214	281	23
Naive	4188	1110	586	2211	281	0

Table 1: Comparison of answers given by TAPAAL with different techniques enabled, where * signifies the use of our explicit engine. The column headers + and - denote positive and negative results respectively. The unique results are compared to Naive.

We denote configurations using our engine as explicit, where explicit old is the version from [11]. Additionally we label configurations full or naive to signify that either every technique is enabled or disabled.

The experiment shows that query simplification is by far the biggest contributor to answering negative queries. The difference in answers for Full and Naive is around 200 for positive answers and 2.300 for negative answers and query simplification alone provides more than 2.000 of the extra negative answers.

Our current implementation with query simplification is ahead of TAPAAL in negative results, indicating it was indeed the primary contributor to the difference in negative answers.

Explicit strategy experimentation Our explicit approach has two different successor generators, namely FIX and EVEN [11]. The difference is their traversal strategy, where FIX explores transition-binding pairs by first iterating through bindings and then transitions s.t. every successor reachable by firing a specific transition is returned in sequential order. When using EVEN the successor generator, if possible, iterates through bindings for a new transition every time it is called, promoting a more varied exploration. Our approach has BFS, DFS, BestFS, and RDFS implemented as search strategies [11], with the difference being the data structure used for the waiting list, where DFS uses a stack, BFS uses a queue, BestFS uses a priority queue with a heuristic distance function, and RDFS uses a stack where the successor generator finds multiple successors which are added in a randomised order. To compare the strategies we experimentally tested each permutation as well as TAPAAL with every technique enabled, which can be seen in Table 2. It is clear that the EVEN successor generator overall performs best with every search strategy answering more

		FIX				EVEN				TAPAAL
		BFS	DFS	BestFS	RDFS	BFS	DFS	BestFS	RDFS	
FIX	BFS	0	**1361**	713	1175	13	1432	741	**1504**	1630
	DFS	101	**0**	145	135	107	84	130	**323**	488
	BestFS	178	**870**	0	677	182	940	101	**998**	1127
	RDFS	75	**295**	112	0	79	344	74	**415**	510
EVEN	BFS	5	**1359**	709	1171	0	1429	735	**1498**	1627
	DFS	90	**2**	133	102	95	0	117	**248**	456
	BestFS	188	**837**	83	621	190	906	0	**959**	1081
	RDFS	47	**126**	76	58	49	133	55	**0**	436
TAPAAL		741	**859**	773	721	746	909	745	**1004**	0

Table 2: Pairwise comparison of our implementation using different search strategies and successor generators, showing unique answers for the column technique.

Explicit techniques	Answers	Positive	Negative	Naive* unique
Full*	7287	4051	3236	2547
Query simplification*	7116	3934	3182	2386
Constrained bindings*	4958	3949	1009	404
Naive*	4787	3956	831	0

Table 3: Comparison of our implemented techniques and unique answers compared to Naive*.

queries when using it. Overall EVEN-RDFS is the best performing combination by a decent margin, with DFS being the best performing when using the FIX successor generator. However, there is no combination that is strictly worse as every strategy gives some unique pairwise answers. Unless otherwise specified the experiments use the EVEN-RDFS strategy as it performs the best.

The impact each of our techniques provide can be seen in Table 3, where it is clear that query simplification is the most impactful technique, providing nearly 50% additional answers, notably almost all of them negative. Our constrained bindings technique also mostly provide new negative answers giving almost 200 additional negative answers. In this experiment both techniques when used alone provide fewer positive results than the naive configuration, which can partially be because of the additional overhead but we attribute this almost entirely to natural variance due to using RDFS. The difference in answers while using both techniques and only query simplification is equivalent to the difference between using none and using only constrained bindings. This as well as the increase in positive and unique answers suggest that the techniques help solve different queries and work complementary.

A cactus plot comparing TAPAAL's unfolding with our old and current explicit approach can be seen in Figure 5. It plots the time spent in solving a query instance for each configuration sorted in ascending order. Queries that are solved by all configurations in less than 0.1 seconds are omitted.

Comparing the old and new engine for our explicit approach, they have nearly the same slope with the new engine being slightly steeper and solving signifi-

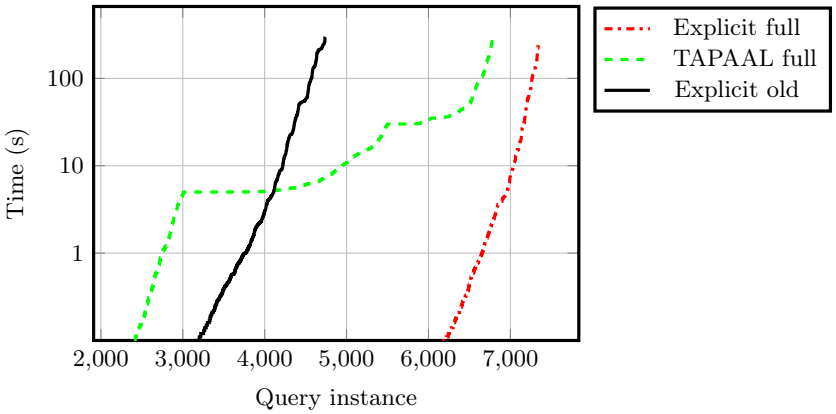


Fig. 5: Cactus plot comparing different configurations.

Over-approximation	Answers	Positive	Negative	Cardinality	Fireability	Unique
Explicit*	**3570**	**580**	**2990**	2861 (141)	**709** (646)	**787**
TAPAAL[17]	2976	262	2714	**2898** (178)	78 (15)	193

Table 4: Comparison of TAPAAL's existing and our over-approximation technique. The best performing technique in each category is highlighted in bold and the unique answers are noted in parenthesis.

cantly more queries nearly instantaneously. This coincides with our other experiments where query simplification solves a significant amount of queries, where our other optimisations provide more answers and answer queries faster but the difference is not nearly as drastic leading to a slightly steeper slope. The overhead of unfolding is also clear, where either explicit approach solve significantly more queries given up to five seconds, where the unfolding approach solves around 2.000 queries. While unfolding is close in the amount of answers found given the full amount of time, the amount of queries able to be solved in one second is nearly three times higher for our new engine.

Query simplification The approach of using over-approximation for query simplification on CPNs is already implemented in TAPAAL [17], which we want to compare to our implementation. In this experiment only answers obtained from query simplification using over-approximation is included and the results can be seen in Table 4. Note that TAPAAL has two different approaches for query simplification where one uses over-approximation and the other first unfolds the net and then uses PN query simplification methods, which is the one used in Table 1.

The results show that our technique provides more answers, but the existing one still has some unique results. The difference in performance is almost completely due to our technique performing much better on fireability queries, while the existing technique performs slightly better on cardinality queries. Allowing

MCC Script	Answers	Cardinality	Fireability
Using explicit	7862 (91%)	4267 (98%)	3595 (83%)
Without explicit	7088 (82%)	4051 (93%)	3037 (70%)

Table 5: Comparison of answers using the script submitted to MCC'25 with and without including our explicit approach, with the percentage of answered queries in the category in parenthesis.

negated fireability expressions in our technique clearly allows for solving significantly more fireability queries. The difference in cardinality queries is most likely due to the difference in how subtraction is handled[2].

Model Checking Competition In the MCC the goal is to solve the greatest number of queries, within the given time limit which is 60 minutes for each query category and model using 4 CPU cores and 16 GiBs of memory. In practice TAPAAL tries different strategies and parameters for the different queries, until either the time runs out or the query has been solved. In the script submitted by TAPAAL for reachability queries to MCC'25 explicit state space exploration is used for 144 seconds with four different strategies in parallel before unfolding is used instead. Some different combinations of strategies were tried with the best result achieved by using EVEN-BFS, EVEN-DFS, FIX-HEUR, and EVEN-RDFS, using query simplification only with EVEN-RDFS. We experimentally used the script for the MCC'24 dataset with and without our explicit approach with the results shown in Table 5.

Using our explicit approach provides a decent increase in cardinality queries answered and a very significant increase for fireability queries of almost 20%, which combined solves almost exactly half of the missing queries with only 85 cardinality queries remaining unanswered.

6.3 LTL experimentation

Using our explicit engine on the LTL queries gives the results seen in Table 6. Overall our approach performs poorly compared to TAPAAL and although there is a decent amount of unique answers the performance is worse for both positive and negative answers. Interestingly, unlike reachability, we perform comparatively better on negative queries, where TAPAAL NDFS is around 15% ahead, than for positive queries where it is around 26% ahead. It is also clear that Tarjan performs better than NDFS for positive queries, but the difference between TAPAAL NDFS and Tarjan is still smaller than the difference between TAPAAL NDFS and our approach. We experimentally tested using both our EVEN and FIX successor generator, but there was no noticeable difference. We think the reason for this poor performance is related to our implementation which does

[2] There is a documented bug in TAPAAL's over-approximation when arcs include subtraction which in some cases cause it to under-approximate. While it has not caused any inconsistencies in our experiments, it could be a factor in some unique results.

LTL Techniques	Answers	Positive	Negative	Unique
TAPAAL Tarjan	7088	4816	2272	0
TAPAAL NDFS	6641	4377	2264	0
Explicit NDFS*	5442	3477	1965	370

Table 6: Comparison of techniques for LTL queries, showing unique answers compared to the combined answers of TAPAAL NDFS and TAPAAL Tarjan.

not include many optimisations and is overall not as efficient as our reachability implementation.

7 Conclusion

We presented an extension to our explicit state space exploration for CPNs which allows it to solve LTL queries, as well as improvements for reachability queries. The techniques introduced for reachability queries are query simplification by over-approximation and constraining the bindings generated by analysing the model and current marking. We proved their correctness and using models and queries from MCC'24 showed their efficacy experimentally as well as comparing our results with the tool TAPAAL which following the current state of the art uses unfolding. Our previous implementation performed slightly better than TAPAAL for reachability queries with a counterexample and significantly worse for those without, whereas now, primarily because of query simplification, our explicit state space exploration approach answers more queries of both types than TAPAAL while being a significant improvement for fireability queries. This suggests that an explicit state space exploration approach is better than the current state of the art approach of unfolding when solving reachability queries for CPNs. Using our explicit state space exploration approach for LTL queries does not perform as well as TAPAAL in any category, however it still provides answers to 370 unique queries that neither of TAPAAL's LTL strategies can answer.

8 Future work

In our query simplification when creating a color ignorant net, we create new transitions for every value between *min-cardinality* and *max-cardinality*, although this approach retains all behaviour, it could be optimised further by only using the possible values. However, we assume the difference in practice to be minimal.

In our query simplification, we either return an answer if the query can be fully simplified or perform state space exploration on the original query if it cannot. Instead the query simplification could evaluate each sub expression in the query replacing it with *true* or *false* in the original query if it can be simplified. This form of query reduction allows the simplification technique to be beneficial for a larger amount of queries. Especially fireability queries could benefit as checking whether a transition is enabled in a marking is relatively time consuming compared to checking cardinality.

Currently, bindings from color constraints are generated for variables that only appear in outgoing arcs. Since the value of those variables do not affect whether the transition is enabled, a possible optimisation would be to only iterate through those bindings, if the transition is enabled when disregarding those variables.

Implementing partial order reductions, such as stubborn reductions, could improve performance since it is effective for uncolored nets, providing about 400 extra answers over naive unfolding.

The LTL implementation could be improved and extended, e.g. implement Tarjan or implement some form of query-reduction or -simplification. Support for more query types such as CTL could be implemented, however, investigating why LTL performs worse than we would expect from the reachability results would perhaps be a better first step.

When evaluating fireability queries the current implementation generates the next state with the successor generator and then independently checks whether the transitions in the query are enabled, resulting in some transitions being checked twice. An improvement could be made where the enabledness of transitions are shared between the query checker and the successor generator.

Currently our testing has mostly consisted of using the MCC'24 dataset, which consists of many models and queries providing a relatively high confidence in the correctness of the implementation, however the dataset does not include models with inhibitor arcs or arcs with nested subtraction. This means that the implementation of those features are not as thoroughly tested. Creating models with these characteristics to test the implementation would help find potential bugs and provide a higher confidence in the correctness of the implementation.

Acknowledgments. We want to thank Jiří Srba for his excellent supervision and Peter Gjøl Jensen for technical assistance.

References

1. Amat, N., Amparore, E., Berthomieu, B., Bouvier, P., Zilio, S.D., Hulin-Hubard, F., Jensen, P.G., Jezequel, L., Kordon, F., Li, S., Paviot-Adet, E., Petrucci, L., Srba, J., Thierry-Mieg, Y., Wolf, K.: Behind the scene of the model checking contest, analysis of results from 2018 to 2023. In: Beyer, D., Hartmanns, A., Kordon, F. (eds.) TOOLympics Challenge 2023. pp. 52–89. Springer Nature Switzerland, Cham (2025)
2. Baier, C., Katoen, J.P.: Principles of Model Checking, vol. 26202649 (01 2008)
3. Bilgram, A., Jensen, P.G., Pedersen, T., Srba, J., Taankvist, P.H.: Improvements in unfolding of colored petri nets. In: Bell, P.C., Totzke, P., Potapov, I. (eds.) Reachability Problems. pp. 69–84. Springer International Publishing, Cham (2021)
4. Bønneland, F., Dyhr, J., Jensen, P.G., Johannsen, M., Srba, J.: Simplification of ctl formulae for efficient model checking of petri nets. In: Khomenko, V., Roux, O.H. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 143–163. Springer International Publishing, Cham (2018)
5. Bønneland, F., Dyhr, J., Jensen, P., Johannsen, M., Srba, J.: Stubborn versus structural reductions for petri nets. Journal of Logical and Algebraic Methods in Programming **102** (09 2018). https://doi.org/10.1016/j.jlamp.2018.09.002

6. Courcoubetis, C., Vardi, M., Wolper, P., Yannakakis, M.: Memory-efficient algorithms for the verification of temporal properties. Formal Methods in System Design **1**(2), 275–288 (Oct 1992). https://doi.org/10.1007/BF00121128, https://doi.org/10.1007/BF00121128
7. Courcoubetis, C., Vardi, M., Wolper, P., Yannakakis, M.: Memory-efficient algorithms for the verification of temporal properties. Formal Methods in System Design **1**, 275–288 (10 1992). https://doi.org/10.1007/BF00121128
8. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K., Møller, M., Srba, J.: TAPAAL 2.0: integrated development environment for timed-arc Petri nets. In: Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12). LNCS, vol. 7214, p. 492–497. Springer-Verlag (2012)
9. van Diepen, J.A., Jensen, N.Ø., Sørensen, M.M.: Structural reductions of colored petri nets and p/t nets with inhibitor arcs (2022), https://kbdk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/a7me0f/alma9921568344605762
10. Geldenhuys, J., Valmari, A.: More efficient on-the-fly ltl verification with tarjan's algorithm. Theoretical Computer Science **345**(1), 60–82 (2005). https://doi.org/https://doi.org/10.1016/j.tcs.2005.07.004, https://www.sciencedirect.com/science/article/pii/S0304397505003853, tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)
11. Højriis, J.E.F., Brandt, E.N.: Explicit state space exploration for colored petri nets (2025), https://kbdk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/a7me0f/alma9921966402705762, pre-specialisation student project
12. Højriis, J.E.F., Brandt, E.N., Stæhr Pedersen, K.: Cs-24-sv-9-04 / verifypn (2025), https://github.com/CS-24-SV-9-04/verifypn
13. Jensen, K.: Coloured petri nets and the invariant-method. Theoretical Computer Science **14**(3), 317–336 (1981). https://doi.org/10.1016/0304-3975(81)90049-9, https://www.sciencedirect.com/science/article/pii/0304397581900499
14. Jensen, P.G., Srba, J., Ulrik, N.J., Virenfeldt, S.M.: Automata-driven partial order reduction and guided search for ltl model checking. In: Finkbeiner, B., Wies, T. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 151–173. Springer International Publishing, Cham (2022)
15. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: Ltsmin: High-performance language-independent model checking. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 692–707. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
16. Klostergaard, A.H.: Efficient Unfolding and Approximation of Colored Petri Nets with Inhibitor Arcs. Master's thesis, Department of Computer Science, Aalborg University (2018), https://projekter.aau.dk/projekter/files/281079031/main.pdf
17. Klostergaard, A.H.: Efficient Unfolding and Approximation of Colored Petri Nets with Inhibitor Arcs. Master's thesis, AAU (2018), https://projekter.aau.dk/projekter/files/281079031/main.pdf
18. Kordon, F., Bouvier, P., Garavel, H., Hulin-Hubard, F., Jezequel, L., Nivon, E.P.A.Q., , Amat., N., Berthomieu, B., Dal Zilio, S., Jensen, P., Morard, D., Smith, B., Srba, J., Thierry-Mieg, Y., Wolf, K.: Complete Results for the 2024 Edition of the Model Checking Contest. https://mcc.lip6.fr/2024/results.php (June 2024)
19. Liebke, T.: Büchi-automata guided partial order reduction for ltl. In: PNSE@Petri Nets (2020), https://api.semanticscholar.org/CorpusID:220323241

20. MCC: Results for LTLCardinality (Apr 2025), https://mcc.lip6.fr/2024/index.php?CONTENT=results/ReachabilityCardinality.html&TITLE=Results%20for%20ReachabilityCardinality
21. MCC: Results for LTLFireability (Apr 2025), https://mcc.lip6.fr/2024/index.php?CONTENT=results/LTLFireability.html&TITLE=Results%20for%20LTLFireability
22. MCC: Results for ReachabilityCardinality (Apr 2025), https://mcc.lip6.fr/2024/index.php?CONTENT=results/LTLFireability.html&TITLE=Results%20for%20ReachabilityFireability
23. MCC: Results for ReachabilityCardinality (Apr 2025), https://mcc.lip6.fr/2024/index.php?CONTENT=results/LTLCardinality.html&TITLE=Results%20for%20ReachabilityFireability
24. Petri, C.A.: Kommunikation mit automaten (1962), https://api.semanticscholar.org/CorpusID:117254333
25. Somenzi, F., Bloem, R.: Efficient büchi automata from ltl formulae. vol. 1855 (05 2000). https://doi.org/10.1007/10722167_21
26. Thierry-Mieg, Y.: Symbolic model-checking using its-tools. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 231–237. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
27. Wolf, K.: Petri net model checking with lola 2. In: Khomenko, V., Roux, O.H. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 351–362. Springer International Publishing, Cham (2018)