Improvements to Structural Reductions of Petri Nets with Inhibitor Arcs

Jesper A. van Diepen, Nicolaj Ø. Jensen, and Mathias M. Sørensen

Cassiopeia, Department of Computer Science, Aalborg University, Denmark

Abstract Structural reductions of Petri net allow for more efficient model checking and can reduce the state space explosion problem. Previous work formally describes, proves, and implements structural reduction rules in TAPAAL. We extend this work by describing, proving, and implementing generalisations of existing rules, particularly concerning weights and inhibitor arcs, and by introducing new rules including a more general post agglomeration rule. For some of these new rules we show how they can supersede previously presented rules by other authors. Additionally, we fix implementation issues with two existing rules in TAPAAL not matching their specifications.
We experimentally show the impact of our rules on the reduce and verification time of TAPAAL using the Model Checking Competition 2021 dataset. We find that some rules improve performance slightly, and that others significantly reduce the state space size. All new rules are applicable on Petri nets with inhibitor arcs, and combined greatly improve performance on a modified test suite containing inhibitor arcs.

Keywords: *Petri net, structural reductions, post agglomerations, inhibitor arcs*

1 Introduction

Model checking [6] is a formal method to verifying properties in a model of one or more systems and processes, e.g. a transmission protocol, the activation of an airbag, or a traffic light. For a traffic light, important properties are that *all lanes eventually have green light* or that *two crossing lanes do not have a green light at the same time* among others. Model checking is especially useful in critical systems, where reliability and correctness are a priority. When model checking we search the reachable states in the model until we can verify that the model satisfies the property. The main difficulty in model checking, therefore, lies in the size of the state space. Model checking, unfortunately, suffers from the *state space explosion* problem [6], especially when modelling concurrent processes, as all the possible interleavings of these introduce a large number of states, that has to be explored.

One way of reducing the state space explosion is by using structural reductions [6, 21, 5]. The idea in this approach is that we analyse the structure of the given model to remove redundant parts of the model with respect to the property that we wish to verify. Some structural reductions lead to an exponential reduction in the state space size while others simply make the net smaller, saving us processing resources and memory usage.

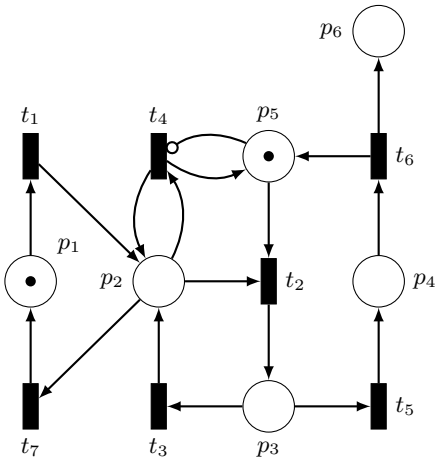
Petri nets [19] are a modelling formalism that can be used to model concurrent systems, and many extensions exist incorporating time [12], types [15], and games [9]. Petri nets have gained popularity due to their graphical nature and their intuitiveness for modelling concurrent processes. In this paper we focus on Petri nets with inhibitor arcs, applying structural reduction rules on Petri nets with inhibitor arcs, and the testing of these structural reductions in the model checking tool TAPAAL [8].

An example of a Petri net can be seen in Figure 1a using the standard graphical notation with squares for transitions, circles for places, dots for tokens, arrows for arcs, and arrows with a circle head-tip for inhibitor arcs. We check if there is a state where the place p_6 contains exactly one token, expressed by the property $\varphi \equiv EFp_6 = 1$. Figure 1b shows the state space of reachable markings from the initial marking. The notation in the square represents the marking of the state, e.g $p_1 + p_5$ means that there is one token in p_1 and one in p_5. For this example, the state space is not particularly large to show a point, but the state space is typically exponentially bigger than the original Petri net.

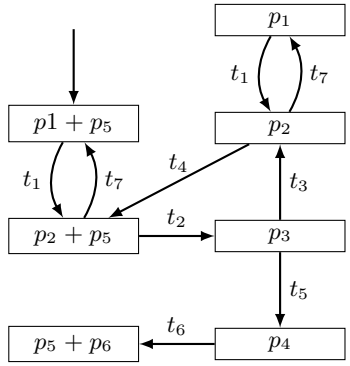
In Figure 1c we apply the structural reduction rules from TAPAAL, as well as the rules introduced in this paper on the Petri net in Figure 1a. The structural reductions are applied w.r.t. to the property φ, which means the reductions preserve the reachability of the goal markings. The state space of this reduced model is shown in Figure 1d. As seen, we end up with a smaller state space to explore, which is the goal of using structural reductions.

1.1 Our contributions

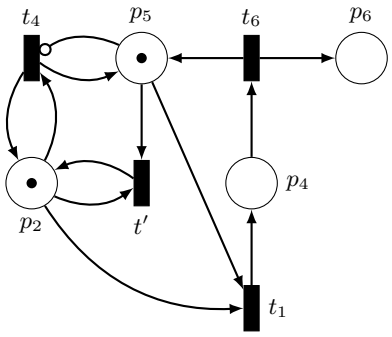
We present further work on the structural reductions of TAPAAL presented by Bønneland et al. in [5]. We fix implementation errors of TAPAAL that made rules more restrictive than intended, i.e less applicable than the specifications given in [5]. We also introduce new extensions to the existing rules, allowing the rules to be applicable in more models, in particular when they include inhibitor arcs. Furthermore, we introduce, implement, and formally prove four new rules. One of these new rules, Rule R, is a post agglomeration more general than the usual definition, agglomerating one producer at a time. Another new rule, Rule M, finds and removes most of the dead parts of a net using fixed-point iteration. Rule R and Rule M supersedes some of the existing rules in [5]. The rest of our new rules are inspired by existing rules presented in [21] by Thierry-Mieg. We contribute by extending the rules to inhibitor arcs and in some cases by relaxing the preconditions and by providing formal proof for the rules. Lastly,



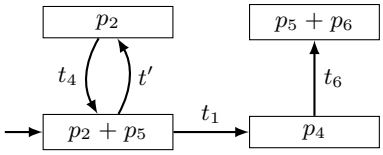
(a) A Petri net. The arc from p_5 to t_4 is an inhibitor arc. All arcs have the default weight 1.



(b) The reachable state space from the initial marking for the Petri net in 1a



(c) The Petri net from Figure 1a after applying the implemented structural reduction rules in TAPAAL



(d) The reachable state space from the initial marking for the Petri net in 1c

Figure 1: (a) Petri net (b) State space of the Petri Net (c) Reduced Petri net (d) State space of reduced Petri net

we implement the rules in TAPAAL and test if our rule fixes, extensions and new rules improve the performance of the tool.

1.2 Related work

Some of the first work studying structural reductions on Petri Net is by Berthelot in [2, 1] focusing on preserving liveness and boundedness. Additional classical structural reductions and related techniques are covered by Murata in [17], including state equations and marked graphs. This work is the foundation of

the structural reductions in the model checking tool TAPAAL as presented by Bønneland et al. in [5]. In the paper Bønneland et al. present new rules with support for inhibitor arcs and also show how structural reductions and stubborn reductions complement each other. The work presented in this paper is further development of the structural reductions in TAPAAL and thus leans heavily on the work by Bønneland et al. In particular, we fix and extend rules by Bønneland et al. to handle inhibited transitions and redundant arcs. In [21] by Thierry-Mieg structural reductions are combined with under- and over-approximations using random runs, SMT solvers, and a counterexample-guided abstraction refinement framework. Recently, agglomerations, which is a particular type of structural reduction, have been studied by Hadded et al. in [10, 11]. Agglomerations focus on removing interleavings in the underlying state space instead of reducing the size of the net itself. We present a new rule, Rule R, that generalises post-agglomerations and works for arbitrary weights.

1.3 Paper outline

In Section 2 we introduce the notation and semantics that we use in the rest of the paper. In Section 3 we first in Section 3.1 show our fixes to TAPAAL, then in Section 3.2 we show and prove our extensions to the existing rules, and finally we introduce and prove our new rules in Section 3.3. Following, in Section 4, we show the experimental results from testing the performance of our proposed fixes, extensions, and new rules. We discuss unexpected results and future work in Section 5. Lastly, in Section 6, we conclude on our findings. An overview of all relevant rules discussed in this paper, [5], and [21] can be found in Appendix A.

2 Preliminaries

Definition 1 (Labeled Transition System). *A labeled transition system (LTS) is a triple* $TS = \langle \mathcal{S}, A, \rightarrow \rangle$ *where*

- $\mathcal{S}$ *is a non-emtpy set of states,*
- A *is a set of actions, and*
- $\rightarrow \subseteq \mathcal{S} \times A \times \mathcal{S}$ *is a transition relation.*

We write $s \xrightarrow{a} s'$ whenever $\langle s, a, s' \rangle \in \rightarrow$. We write $s \xrightarrow{a}$ whenever $\langle s, a, s'' \rangle \in \rightarrow$ for some $s'' \in S$ and say that a is *enabled* in s. We write $s \rightarrow s'$ if there exists an action $a \in A$ and a state $s' \in \mathcal{S}$ such that $s \xrightarrow{a} s'$, and $s \not\rightarrow$ if no such action a exists. For a state s where $s \not\rightarrow$ we say that it is a *dead* state or in a *deadlock*. We extend the relation $\xrightarrow{a}$ inductively to traces $w \in A^*$ such that $s \xrightarrow{\varepsilon} s$ and $s \xrightarrow{wa} s'$ if $s \xrightarrow{w} s''$ and $s'' \xrightarrow{a} s'$. We write $s \rightarrow^n s'$ if there exists a $w \in A^n$ such that $s \xrightarrow{w} s'$, and we write $s \rightarrow^* s'$ if $s \rightarrow^n s'$ for some $n \geq 0$. If $w \in A^\omega$ it is an infinite trace. Given a set X we let $X^\bigstar = X^* \cup X^\omega$ be all finite and infinite sequences of elements from X. We say that a is *fireable* if s_0 is the initial state of an LTS and $s_0 \rightarrow^* s' \xrightarrow{a}$.

2.1 Petri nets with inhibitor arcs

Let $\mathbb{N}^0 = \mathbb{N} \cup \{0\}$ be the natural numbers including 0, and let $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$ be the natural numbers including ∞.

Definition 2 (Petri net with inhibitor arcs). *A Petri net with inhibitor arcs is a 5-tuple* $N = \langle P, T, W_\boxminus, W_\boxplus, I \rangle$ *where*

- P *is a finite non-empty set of places,*
- T *is a finite set of transitions,*
- $W_\boxminus : P \times T \to \mathbb{N}^0$ *and* $W_\boxplus : P \times T \to \mathbb{N}^0$ *are pre- and post incidence matrices, and*
- $I : P \times T \to \mathbb{N}^\infty$ *is an inhibitor weight matrix.*

A marking $M : P \to \mathbb{N}^0$ on N is a function (but sometimes we use it as a vector), where $M(p)$ denotes the number of tokens at place $p \in P$. The set $\mathcal{M}(N)$ contains all markings of a Petri net N. The effect matrix $E : P \times T \to \mathbb{Z}$ describes how the firing of transition t affects place p. That is $E(p,t) = W_\boxplus(p,t) - W_\boxminus(p,t)$ for all $p \in P$ and $t \in T$. We write $W_\boxminus(t)$, $W_\boxplus(t)$, $I(t)$, and $E(t)$ for a given transition t to denote a column vector with $|P|$ entries. Given two vectors v, v' and comparison operator $\bowtie \in \{<, \leq, =, \geq, >\}$ we write $v \bowtie v'$ when $v(i) \bowtie v'(i)$ for all entries i.

Additional notation that we define:

- the *pre set* (producers) of a place $p \in P$ as ${}^\bullet p = \{t \in T \mid W_\boxplus(p,t) > 0\}$,
- the *post set* (consumers) of a place $p \in P$ as $p^\bullet = \{t \in T \mid W_\boxminus(p,t) > 0\}$,
- the *pre set* of a transition $t \in T$ as ${}^\bullet t = \{p \in P \mid W_\boxminus(p,t) > 0\}$,
- the *post set* of a transition $t \in T$ as $t^\bullet = \{p \in P \mid W_\boxplus(p,t) > 0\}$,
- the *increasing pre set* for a place $p \in P$ as ${}^\boxplus p = \{t \in {}^\bullet p \mid E(p,t) > 0\}$,
- the *descreasing post set* for a place $p \in P$ as $p^\boxminus = \{t \in p^\bullet \mid E(p,t) < 0\}$,
- the *inhibiting pre set* for a transition $t \in T$ as ${}^\circ t = \{p \in P \mid I(p,t) < \infty\}$,
- the *inhibiting post set* for a place $p \in P$ as $p^\circ = \{t \in T \mid I(p,t) < \infty\}$.

For a set X of either places or transitions, we extend the notation as ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$, and likewise for similar operators.

A Petri net $N = \langle P, T, W_\boxminus, W_\boxplus, I \rangle$ defines an LTS $TS(N) = \langle \mathcal{S}, A, \to \rangle$, where $\mathcal{S} = \mathcal{M}(N)$, and $A = T$, and $M \xrightarrow{t} M'$ whenever $W_\boxminus(t) \leq M < I(t)$ where $M' = M + E(t)$. The *initial marking* is denoted M_0.

2.2 CTL* and its subclasses

In this section we introduce the formalism *computation tree logic star* (CTL*) which is a superset of *computation tree logic* (CTL) and *linear temporal logic* (LTL). We also cover these two logics, as well as *reachability* (EF/AG) and *deadlock properties*. The relationship between these logics can be seen in Figure 2.

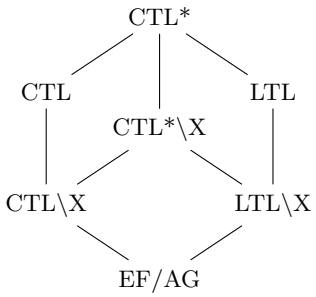


Figure 2: Logic formalisms and their relationship in a lattice.

2.2.1 CTL* There are two types of CTL* formulas: *state formulas* and *path formulas*. Formally, a CTL* formula has one of the following forms:

$$\varphi ::= \top \mid \pi \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid A\psi_1 \mid E\psi_1$$
$$\psi ::= \varphi_1 \mid \neg\psi_1 \mid \psi_1 \wedge \psi_2 \mid X\psi_1 \mid F\psi_1 \mid G\psi_1 \mid [\psi_1 U \psi_2]$$

where φ is the state formulas, ψ is the path formulas, and $\pi \in \Pi$ is an atomic proposition. In the context of Petri nets where states are markings, such a proposition π has the form $\alpha M \bowtie k$ where M is the given marking, $\alpha \in \mathbb{R}^P$ is a vector of coefficients, and $k \in \mathbb{R}$ a constant, and $\bowtie \in \{<, \leq, =\geq, >\}$ is a comparison operator. Often we write αM as a linear equation using the place names to represent the number of tokens at that place, e.g. $2p_1 + p_2 \leq 5$. Two special propositions are $en(t)$, that asserts that the transition $t \in T$ is enabled, and *deadlock*, that asserts that no transitions are enabled. Furthermore, A and E are *path quantifiers*, and X, F, G, U are *temporal operators*. Additional boolean and temporal operators can be derived.

Remark 1. The proposition $en(t)$ can be rewritten to a conjunction of atomic propositions. This allows us to check fireability of t, even if t is removed from the net by structural reductions.

In the semantics below, we let $\lambda = \mathcal{S}^\bigstar$ denote a path such that $\lambda_1 \to \lambda_2 \to \ldots$ and let $\lambda[i]$ denote the sub-path of λ starting from λ_i. A *maximal path* λ' is either infinite or ends in a deadlock. For state formulas the semantics are defined inductively such that $s \vDash \varphi$ denotes a state s satisfying the state formula φ.

$s \vDash \top \quad \equiv \top$
$s \vDash \pi \quad \equiv \pi(s)$
$s \vDash \neg\varphi \quad \equiv s \nvDash \varphi$
$s \vDash \varphi_1 \wedge \varphi_2 \equiv (s \vDash \varphi_1) \wedge (s \vDash \varphi_2)$
$s \vDash A\psi \quad \equiv \lambda \vDash \psi$ for all maximal paths λ starting in s
$s \vDash E\psi \quad \equiv \lambda \vDash \psi$ for some maximal path λ starting in s

Path formulas are also defined inductively, such that $\lambda \vDash \psi$ denotes path λ satisfying path formula ψ.

$$
\begin{array}{lcl}
\lambda \vDash \varphi & \equiv & \lambda_1 \vDash \varphi \\
\lambda \vDash \neg\psi & \equiv & \lambda \nvDash \psi \\
\lambda \vDash \psi_1 \wedge \psi_2 & \equiv & (\lambda \vDash \psi_1) \wedge (\lambda \vDash \psi_2) \\
\lambda \vDash X\psi & \equiv & \lambda[1] \vDash \psi \\
\lambda \vDash F\psi & \equiv & \lambda[n] \vDash \psi \text{ for some } n \geq 0 \\
\lambda \vDash G\psi & \equiv & \lambda[n] \vDash \psi \text{ for all } n \geq 0 \\
\lambda \vDash \psi_1 U \psi_2 & \equiv & (\lambda[k] \vDash \psi_1) \wedge (\lambda[n] \vDash \psi_2) \text{ for some } n \geq 0, \text{ for all } k, 0 \leq k < n
\end{array}
$$

2.2.2 CTL CTL is a proper subset of CTL* because it only describes properties of branching systems. This is reflected in the syntax, specifically, all temporal operators must be preceded by a path quantifier. CTL formulas have the following form:

$$
\begin{aligned}
\varphi ::= \top \mid \pi \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid AX\varphi_1 \mid AF\varphi_1 \mid AG\varphi_1 \mid A[\varphi_1 U \varphi_2] \mid \\
EX\varphi_1 \mid EF\varphi_1 \mid EG\varphi_1 \mid E[\varphi_1 U \varphi_2]
\end{aligned}
$$

The semantics of the operators are unchanged.

2.2.3 LTL LTL is another proper subset of CTL*. This subset of CTL* describes properties of traces and uses no path quantifiers, except for an implicitly prefixed A operator. LTL formulas have the following form:

$$
\psi ::= \pi \mid \neg\psi_1 \mid \psi_1 \wedge \psi_2 \mid X\psi_1 \mid F\psi_1 \mid G\psi_1 \mid [\psi_1 U \psi_2]
$$

There are two types of semantics for LTL. The first semantic is the one presented earlier for CTL*, and the alternative LTL semantics considers only infinite traces. Traces that normally end in a deadlock are extended by letting the system repeat the deadlock state infinitely, i.e. the system stutters infinitely. The Model Checking Competition [16] use the alternative semantics. The structural reductions presented in this paper work for both semantics.

2.2.4 Stuttering To proceed, we first define stuttering and stutter equivalence. Given a set of atomic propositions Π and a labeling function $L : \mathcal{S} \to 2^{\Pi}$, we can label each state with a subset of propositions, which are true in the given state. A transition system is said to *stutter* when it moves from one state to another state where the same subset of the atomic propositions are true, i.e. if $s \to s'$ and $L(s) = L(s')$.

Definition 3 (Stutter equivalence). *Let Π be a set of atomic propositions, and let $\sigma, \sigma' \in (2^{\Pi})^{\star}$ be two sequences of subsets of propositions such that $\sigma = \rho_0, \rho_1, \rho_2, \ldots$ and $\sigma' = \rho'_0, \rho'_1, \rho'_2, \ldots$. Then σ and σ' are said to be* stutter equivalent *iff there exists two infinite sequences $0 = i_0 < i_1 < \ldots$ and $0 = j_0 < j_1 < \ldots$ such that for all $k \geq 0$ we have $\rho_{i_k} = \rho_{i_k+1} = \cdots = \rho_{i_{k+1}-1} = \rho'_{j_k} = \rho'_{j_k+1} = \cdots = \rho'_{j_{k+1}-1}$*

Given an LTS $\langle \mathcal{S}, A, \rightarrow \rangle$, a set of atomic propositions Π, and a labeling function $L : \mathcal{S} \rightarrow 2^{\Pi}$, a path $\lambda \in \mathcal{S}^{\bigstar}$ defines a sequence of subsets of propositions $L(\lambda_1), L(\lambda_2), \ldots$. Hence we say that two paths $\lambda, \lambda' \in \mathcal{S}^{\bigstar}$ are stutter equivalent w.r.t. L if their underlying sequences of subsets of propositions are stutter equivalent. Similarly, a trace $w \in A^{\bigstar}$ where $s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} s_2 \ldots$ for some initial state $s_0 \in \mathcal{S}$ defines a path visiting states $s_0, s_1, \ldots$. Hence, we say that two traces $w, w' \in A^{\bigstar}$ are stutter equivalent w.r.t. L if the paths that they visit are stutter equivalent w.r.t. L.

Properties that use the temporal next operator X are called stutter-sensitive properties. Not all reductions are applicable when checking stutter-sensitive properties and these reductions are therefore called stutter insensitive [18].

We use CTL*\X, CTL\X, and LTL\X to denote stutter-insensitive CTL*, CTL, and LTL without the next operator X. Clearly, these are respectively proper subsets of CTL*, CTL, and LTL.

Definition 4 (Stuttering bisimilarity). *[3, 6] Given two LTSs $TS = \langle \mathcal{S}, A, \rightarrow \rangle$ and $TS' = \langle \mathcal{S}', A', \rightarrow' \rangle$ with initial states s_0 and s'_0, TS and TS' are said to be stuttering bisimilar w.r.t some set Π of atomic propositions and two labeling functions $L : \mathcal{S} \rightarrow 2^{\Pi}$ and $L' : \mathcal{S}' \rightarrow 2^{\Pi}$, if there exists a stuttering bisimulation $\mathcal{R}$ such that $s_0 \mathcal{R} s'_0$. A stuttering bisimulation $\mathcal{R}$ is an equivalence relation on $\mathcal{S} \times \mathcal{S}'$ such that whenever $s_1 \mathcal{R} s'_1$ then:*

1. *$L(s_1) = L'(s'_1)$,*
2. *for each maximal path $\lambda = s_1 s_2 \cdots \in \mathcal{S}^{\bigstar}$ where $s_1 \rightarrow s_2 \rightarrow \ldots$ there exists an maximal path $\lambda' = s'_1 s'_2 \cdots \in \mathcal{S}'^{\bigstar}$ where $s'_1 \rightarrow' s'_2 \rightarrow' \ldots$ such that λ and λ' can be partitioned into consecutive subpaths $B_1 B_2 \ldots$ and $B'_1 B'_2 \ldots$, respectively and $s \mathcal{R} s'$ for every s in B_i and s' in B'_i, for all $i > 0$.*
3. *symmetrically, for each path λ' from s'_1 there exists a subpathwise matching path λ from s_1.*

Theorem 1 (Stutter invariance). *[3] Let s and s' be two states. If $s \mathcal{R} s$ for some stuttering bisimulation $\mathcal{R}$, then $s \vDash \varphi$ iff $s' \vDash \varphi$ for every CTL*\X formula φ.*

Theorem 2 (LTL stutter invariance). *[20] Let $\sigma, \sigma' \in (2^{\Pi})^{\bigstar}$ be two stutter equivalent sequences of subsets of propositions, and let ψ be an LTL\X formula. Then $\sigma \vDash \psi$ iff $\sigma' \vDash \psi$.*

2.2.5 Reachability and safety In many cases we are only interested in very simple properties. E.g. that the system eventually reaches some *good state*, or that it never reaches some *bad state*, like a deadlock. These properties are called reachability (EF) and safety (AG), and they are a subset of both CTL and LTL, as they do not have nested temporal operators nor path quantifiers. Note that reachability and safety properties are duals and $EF\varphi \equiv \neg AG \neg \varphi$.

3 Structural reduction rules

To verify properties of systems, verification tools use search algorithms to find states and traces that either prove or disprove the property. The performance of these algorithms is primarily affected by the size of the state space and their ability to find traces to a certain subset of states in this state space. As shown in the introduction, structural reductions reduce the size of the state space by transforming the model of the system while preserving the given property.

When performing a structural reduction, we cannot remove places relevant for the atomic propositions of the properties. Therefore, given a CTL* formula φ we use the function $\mathit{places}(\varphi)$ to represent all the places in the Petri net mentioned in the formula φ. Specifically, the function *places* is defined inductively as:

$$
\begin{aligned}
&\mathit{places}(\top) = \emptyset \\
&\mathit{places}(\pi) = \{p \in P \mid \alpha(p) \neq 0\} \quad \text{where } \pi = \alpha M \bowtie k \\
&\mathit{places}(\neg\varphi) = \mathit{places}(\varphi) \\
&\mathit{places}(\varphi_1 \wedge \varphi_2) = \mathit{places}(\varphi_1) \cup \mathit{places}(\varphi_2) \\
&\mathit{places}(A\psi) = \mathit{places}(\psi) \\
&\mathit{places}(E\psi) = \mathit{places}(\psi) \\
&\mathit{places}(\neg\psi) = \mathit{places}(\psi) \\
&\mathit{places}(\psi_1 \wedge \psi_2) = \mathit{places}(\psi_1) \cup \mathit{places}(\psi_2) \\
&\mathit{places}(X\psi) = \mathit{places}(\psi) \\
&\mathit{places}(F\psi) = \mathit{places}(\psi) \\
&\mathit{places}(G\psi) = \mathit{places}(\psi) \\
&\mathit{places}([\psi_1 U \psi_2]) = \mathit{places}(\psi_1) \cup \mathit{places}(\psi_2)
\end{aligned}
$$

Definition 5 (Invisibility). *If φ is a stutter insensitive formula (without the next operator X), a transition $t \in T$ is said to be* invisible *to φ if for all $p \in \mathit{places}(\varphi)$ we have that $E(p,t) = 0$, since the transition t does not affect the satisfaction of the atomic propositions in φ.*

Following in this section, we first discuss implementation fixes to Rule E and I of TAPAAL. Then we propose Rule N, O, and P which extend Rule E and F of TAPAAL, by adding support for inhibitor arcs. Lastly, we present four new rules, Rule L, M, Q, and R. For each extension and new rule we provide proof of correctness for a given logic. We define the correctness of a rule below:

Definition 6 (Correctness of Rule X for logic L).
Let $N = \langle P, T, W_{\boxminus}, W_{\boxplus}, I\rangle$ be a Petri net and let $M_0 \in \mathcal{M}(N)$ be its initial marking. Let $N^{\varphi} = \langle P^{\varphi}, T^{\varphi}, W^{\varphi}_{\boxminus}, W^{\varphi}_{\boxplus}, I^{\varphi}\rangle$ with initial marking $M_0^{\varphi} \in \mathcal{M}(N^{\varphi})$ be the Petri net obtained by applying Rule X on N once for some formula φ of logic L. Rule X is correct for logic L whenever $M_0 \vDash \varphi$ in N iff $M_0^{\varphi} \vDash \varphi$ in N^{φ} for all $\varphi \in L$.

When a rule is correct for a class of logic, it is also correct for its subclasses.

3.1 Fixes to existing Rules

During the development of the new and extended rules that are presented in Section 3.2 and Section 3.3, discrepancies were found between some of the theory presented in [5] and its implementation in TAPAAL. Specifically, the implementation of Rule E contained constraints tighter than those described in the paper, causing it to disregard some potential reductions that were already argued to be valid. Additionally, a helper function used in Rule I had an implementation error in its handling of inhibitor arcs. This section briefly describes these discrepancies.

3.1.1 Rule E The specification of Rule E as described in [5] is shown in Figure 3. The rule determines a local upper bound on the number of tokens in a place p_0. Any transition that consumes more tokens from p_0 than the upper bound will never be enabled and can be removed from the net. Additionally, if applying Rule E leaves p_0 with no consuming transitions, $p_0 \notin places(\varphi)$, and p_0 does not have inhibitor arcs, rule E removes p_0 as well.

Precondition	Update
Fix p_0 and t_0 s.t. E1) $M_0(p_0) < W_{\boxminus}(p_0, t_0)$ E2) $W_{\boxplus}(p_0, t) \leq W_{\boxminus}(p_0, t)$ or $M_0(p_0) < W(p_0, t)$ for all $t \in T$	UE1) If $p_0^\bullet = \{t_0\}$, $p_0^\circ = \emptyset$, and $p_0 \notin places(\varphi)$, then remove p_0. UE2) Remove t_0.

Figure 3: Rule E: Dead transition removal. Specification taken from [5].

The old implementation in TAPAAL had two discrepancies from its specification shown in 3. One was a typo in a *less-than* comparison and another was a wrong assumption about weights in the sparse representation of arcs in TAPAAL. Figures 4 and 5 show reductions that the specification of Rule E allows, but which the old implementation did not.

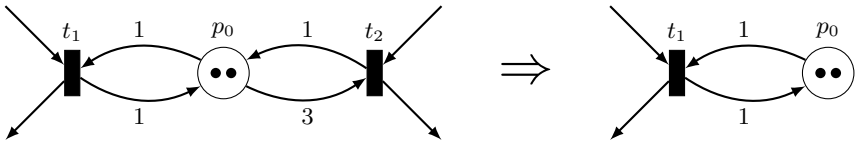


Figure 4: A reduction not possible in the old implementation of Rule E. Previously, an initially enabled transition $t \in p_0^\bullet$ with $0 < W_{\boxminus}(p_0, t) < M_0(p_0)$ and $0 < W_{\boxplus}(p_0, t) \leq W_{\boxminus}(p_0, t)$ would prevent Rule E. Here t_1 is such a t.

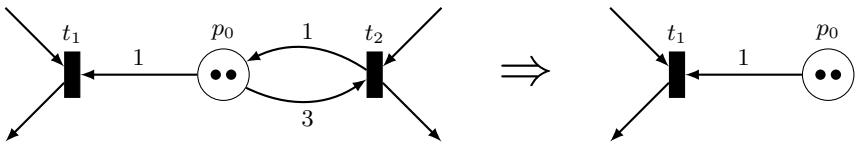
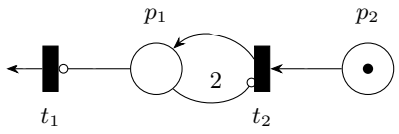


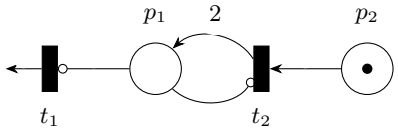
Figure 5: A reductions not possible in the old implementation of Rule E. Previously, enabled consumers of p_0 that returned 0 tokens to p_0 would prevent Rule E.

3.1.2 Rule I The Rule I as described in [5] computes a subset of the model that encompasses an overapproximation of the places and transitions that are relevant to the given query. That is, behaviour is relevant if it affects the places in $places(\varphi)$ or enables other behaviour that is relevant.

TAPAAL's old implementation of Rule I contained issues with its handling of inhibitor arcs. Specifically, the computation of the relevance of parts connected via inhibitor arcs to the part of the model deemed relevant to the query. These issues resulted in excessive overapproximation for unweighted inhibitor arcs, resulting in reductions not being made in some cases where the specification of Rule I allows. Figure 6 shows two model fragments that Rule I can reduce, but only (a) was by the old implementation. In cases involving weighted inhibitor arcs, wrongful reductions of relevant parts of the model were possible, two examples of which are shown in Figure 7.



(a) t_2 and by extension p_2 are not considered relevant since t_2 does not enable relevant behaviour.

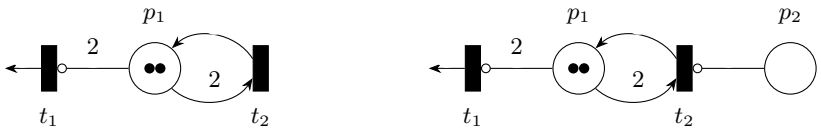


(b) t_2 and by extension p_2 are not considered relevant since t_2 does not enable relevant behaviour. Old implementation of Rule I would keep t_2 and p_2 unnecessarily.

Figure 6: Two model fragments which Rule I should treat identically when $p_1, p_2 \notin places(\varphi)$ and t_1 is relevant, but the old implementation did not.

3.2 Rule extensions

In this section, we present Rules N, O, and P. Rule N is a variation of Rule F presented in [5], that uses the same local lower bound on the number of tokens



(a) Even if t_1 is relevant, t_2 would not be deemed relevant and reduced away, despite affecting the fireability of t_1

(b) A case where the behavior of Rule I was arbitrarily determined by the internal ordering of the arcs, unrelated to the problem in (a).

Figure 7: Two cases of faulty behavior in the old implementation of Rule I, involving weighted inhibitor arcs.

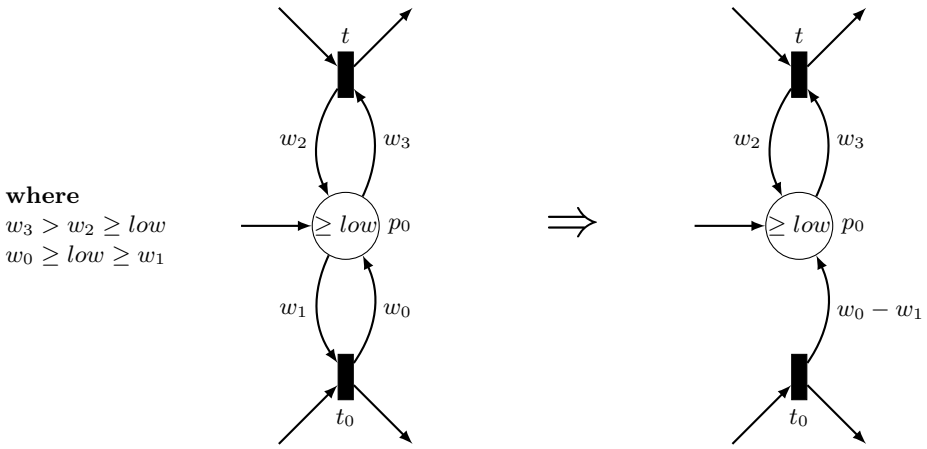
in a place as Rule F, but instead of removing redundant places, it removes individual redundant arcs. While this does not reduce the size of the net, it can allow other rules to be applied in more cases. Rule O also utilizes a local lower bound similar to Rule F and N to remove transitions that can never fire because they are always inhibited. Rule P extends Rule E from [5] to inhibitor arcs. It is the inverse of Rule O, using a local upper bound on tokens in a place to remove inhibitor arcs that are never inhibiting.

The preconditions of Rule F, N, and O are similar and hence their implementations are too. Therefore we recommend that their implementations are combined for better performance. However, to keep the formal descriptions atomic they are presented separately here. Similarly, Rule P should be implemented with Rule E for better performance.

3.2.1 Rule N: Redundant arc removal The lower bound number of tokens at a place p_0 is given by the minimum of the initial marking and the number of tokens returned by any consuming transition with a negative effect on p_0. Using the lower bound we can then find transitions, which are never disabled by p_0 and remove the transition's dependency on p_0, since it is unnecessary, as long as we maintain the effect of firing the transition. A formal description of Rule N is given in Figure 8. Note, that if the weights of the two arcs between p_0 and t_0 are the same, we can also remove the arc from t_0 to p_0, as the updated weight would become 0. While Rule N does not remove places or transitions, it does remove arcs which can allow for other reductions. In particular for rules requiring disjoint pre and post sets such as Rules Q and R presented in Section 3.3. If we end up removing all transitions that depend on p_0 by repeated use of Rule N, then we can apply Rule F and also remove p_0, unless p_0 inhibits a transition, or appears in the query.

Theorem 3. *Rule N in Figure 8 is correct for CTL**

Proof. Let $N = \langle P, T, W_{\boxminus}, W_{\boxplus}, I \rangle$ be a Petri net with initial marking M_0 and let $\varphi \in \mathrm{CTL}^*$. Let N' and M_0' be the net and initial marking after applying Rule N once on N.



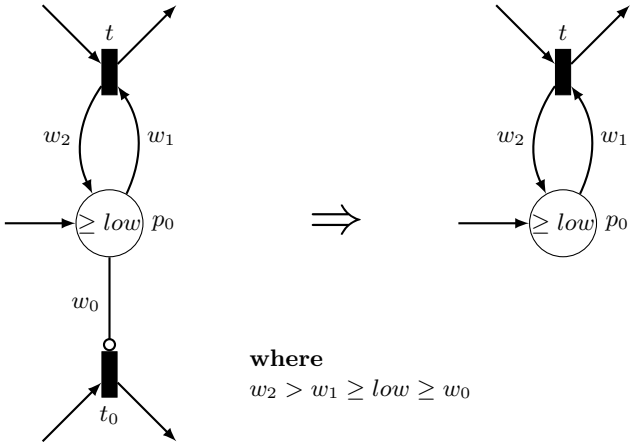
Precondition	Update
Fix place p_0 and transition t_0 s.t.: N1) $t_0 \in p_0^\bullet \setminus p_0^{\boxminus}$ N2) $W_{\boxminus}(p_0, t_0) \leq low$ where $low = \min\{M_0(p_0)\} \cup \{W_{\boxplus}(p_0, t) \mid t \in p_0^{\boxminus}\}$	 UN1) Set $W_{\boxplus}(p_0, t_0) := W_{\boxplus}(p_0, t_0) - W_{\boxminus}(p_0, t_0)$ UN2) Set $W_{\boxminus}(p_0, t_0) := 0$

Figure 8: Rule N: Redundant arc removal

We can find a guarantee of the minimum number of tokens in p_0 denoted *low* in Figure 8. This lower bound is given by either the initial number of tokens $M_0(p_0)$ or by the smallest $W_{\boxplus}(p_0, t)$ for any $t \in p_0^{\boxminus}$, as this represents how many tokens will be left in the place p_0 after removing as many tokens as possible by any transition that removes tokens from p_0. Hence, by $low = \min\{M_0(p_0)\} \cup \{W_{\boxplus}(p_0, t) \mid t \in p_0^{\boxminus}\}$ we know that for all $M \in \mathcal{M}(N)$ where $M_0 \to *M$ we have that $M(p_0) \geq low$. Together with N2 this implies that t_0 is never disabled by p_0.

In N' with the arc removed, the transition t_0 is thus enabled in the exact same markings as in N. By UN1 we have that the effect of t_0 is the same in N and N', i.e. that $E(t_0) = E'(t_0)$. From N1 we know that $W_{\boxminus}(p_0, t_0) \leq W_{\boxplus}(p_0, t_0)$, and therefore that the update UN1 will never be a negative value. Since t_0 is enabled in the same markings in both nets, and since the effect of t_0 is the same, it follows that N and N' is isomorphic. Hence, Rule N is correct. □

3.2.2 Rule O: Inhibited transition As described for Rule N, we can find the lower bound of tokens at a place p_0. Any inhibitor arc from p_0 with a weight smaller than the lower bound always inhibits the given transition, which means that the transition can be removed. See Figure 9 for a formal description of Rule O.



Precondition	Update
Fix place p_0 and transition t_0 s.t.: O1) $t_0 \in p_0^\circ$ O2) $I(p_0, t_0) \leq low$ where $low = \min\{M_0(p_0)\} \cup \{W_\boxplus(p_0, t) \mid t \in p_0^\boxminus\}$	UO1) Remove t_0.

Figure 9: Rule O: Inhibited transition

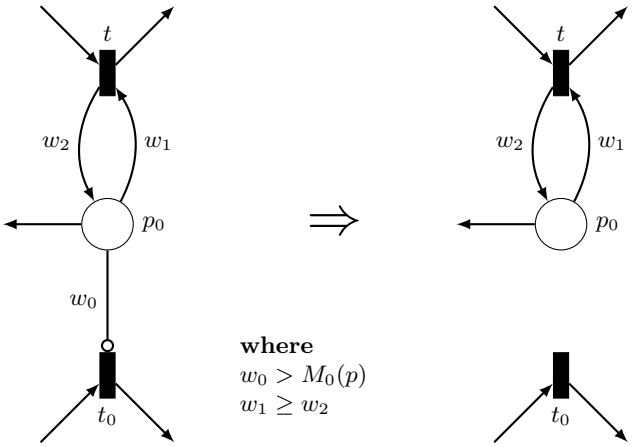
Theorem 4. *Rule O in Figure 9 is correct for CTL**

Proof. Let $N = \langle P, T, W_\boxminus, W_\boxplus, I\rangle$ be a Petri net with initial marking M_0 and let $\varphi \in$ CTL*. Let N' and M_0' be the net and initial marking after applying Rule O once on N.

Similarly to the proof for Theorem 3 for Rule N, we can find a lower bound for the number of tokens in a place p_0 as $low = \min\{M_0(p_0)\} \cup \{W_\boxplus(p_0, t) \mid t \in p_0^\boxminus\}$, such that for all $M \in \mathcal{M}(N)$ where $M_0 \to *M$ we have that $M(p_0) \geq low$. From O1 we know that the transition t_0 is inhibited by p_0. By O2 t_0 will therefore always be inhibited by p_0.

This implies that the reachable state space from M_0' in N' with t_0 removed, is isomorphic to that of M_0, and hence Rule O is correct. □

3.2.3 Rule P: Redundant inhibitor arc In a similar manner to finding the lower bound, we can find an upper bound on the number of tokens at a place p_0. This upper bound is given by the initial marking if all transitions have a non-positive effect on p_0. Any inhibitor arc from p_0 with a weight higher than the upper bound of p_0 therefore never inhibits, which means the inhibitor arc can be removed, potentially making other structural reductions possible. See Figure 10 for a formal description of Rule P.



Precondition	Update
Fix place p_0 and transition t_0 s.t.: P1) $t_0 \in p_0^\circ$ P2) $I(p_0, t_0) > M_0(p_0)$ P3) ${}^{\boxplus}p_0 = \emptyset$	UP1) $I(p_0, t_0) = \infty$.

Figure 10: Rule P: Redundant inhibitor arc

Theorem 5. *Rule P in Figure 10 is correct for CTL*.*

Proof. Let $N = \langle P, T, W_{\boxminus}, W_{\boxplus}, I \rangle$ be a Petri net with initial marking M_0 and let $\varphi \in$ CTL*. Let N' and M_0' be the net and initial marking after applying Rule P once on N. By P3, no transition increases the number of tokens in p_0. Hence the number of tokens in p_0 will at most be equal to $M_0(p_0)$. By P1 and P2, the place

p_0 has an inhibitor arc to the the transition t_0 with a weight $I(p_0, t_0) > M_0(p_0)$. This implies that t_0 never will be inhibited through this arc, and hence by Up1, the reduced net N' has the same behaviour as N. □

3.3 New rules

In this section, we present and prove the correctness of new rules that we implement in TAPAAL. These are the Rules L, M, Q, and R. Rule L and Q are similar to rules by Thierry-Mieg in [21], but we contribute by extending both to inhibitor arcs and generalising Rule Q. We also provide a formal proof for both rules. We also present the novel Rule M and Rule R. Rule M removes effectively dead places and transitions, which Rule R is a more general post agglomeration. We prove correctness of these as well as their ability to supersede existing rules in [5] and [21].

3.3.1 Rule L: Dominated Transition Rule L is described by Thierry-Mieg in [21] as Rule 2. It removes transitions that have the same effect as another transition, but with more preconditions. Since both transitions lead to the same state, we can therefore remove the one with the higher preconditions and use the other instead. In [21] the rule is only defined for normal arcs and not inhibitor arcs. We extend the rule with support for inhibitor arcs by adding the precondition L1 as seen in the formal description in Figure 11.

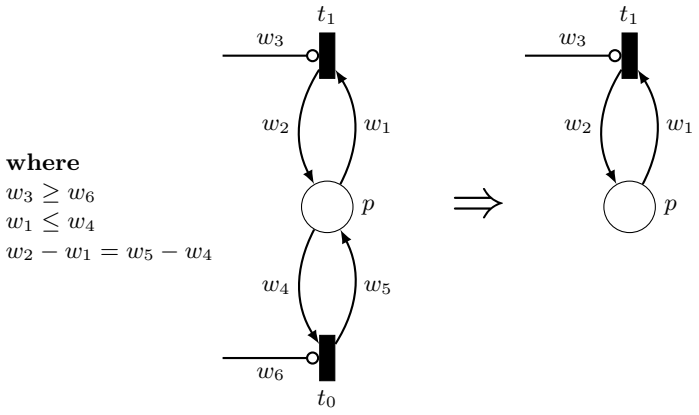
Theorem 6. *Rule L in Figure 11 is correct for CTL*.*

Proof. Let $N = \langle P, T, W_{\boxminus}, W_{\boxplus}, I \rangle$ be a Petri net, and $\varphi \in \text{CTL}^*$. Let N' be the net after applying Rule L once on N. By L1 and L2 t_0 is enabled whenever t_1 is enabled. We know from L3 that we can achieve the effect of t_0 by firing t_1 instead. Removing t_0 from N in UL1 does therefore not change the reachable state space.

3.3.2 Rule M: Effectively dead places and transitions The Rule M finds and removes effectively dead places and transitions. We define an effectively dead place to be a place that will never gain nor lose tokens. Effectively dead transitions are transitions that are initially disabled (and/or inhibited) by a place that cannot gain (and/or lose) tokens. These places and transitions are found using fixed-point iteration as defined in Algorithm 1.

Theorem 7. *Rule M in Algorithm 1 is correct for CTL*.*

Proof. Let $N = \langle P, T, W_{\boxminus}, W_{\boxplus}, I \rangle$ be a Petri net with initial marking M_0 and let $\varphi \in \text{CTL}^*$. Let N' and M'_0 be the net and initial marking after applying Rule M once on N. We will now prove by contradiction that all places removed while using Rule M never gain nor lose tokens and that all transitions removed while using Rule M cannot fire.



Precondition	Update
Fix transition t_1 and t_0 s.t.: L1) $I(t_1) \geq I(t_0)$ L2) $W_{\boxminus}(t_1) \leq W_{\boxminus}(t_0)$ L3) $E(t_1) = E(t_0)$	UL1) Remove t_0

Figure 11: Rule L: Dominated Transition

Assume place p was removed but that there exists a reachable marking M such that $M_0(p) \neq M(p)$ and $w \in T^*$ is the shortest trace such that $M_0 \xrightarrow{w} M$. By line 1-2 of the Algorithm 1, $p \in S_{\leq} \cap S_{\geq}$ initially and note that the algorithm never adds elements to F, $S_{\leq}$, or $S_{\geq}$. Since p was removed, we must have $p \in S_{\leq} \cap S_{\geq}$ too after the loop by line 10. However, we will now show by induction, that for all i, $0 < i \leq |w|$, the transition w_i satisfies the condition in line 5 and is therefore not in F, while $w_i^{\boxplus}$ and ${}^{\boxminus}w_i$ are not in $S_{\leq}$ and $S_{\geq}$, respectively. This implies that $p \notin S_{\leq} \cap S_{\geq}$ creating a contradiction.

- Let $i = 1$. Since $M_0 \xrightarrow{w}$, then w_i must have been initially enabled, which means $W_{\boxminus}(p, w_i) \leq M_0(p)$ and $I(p, w_i) > M_0(p)$ for all $p \in P$. This implies that w_i satisfies the condition in line 5.
- Let $i > 1$. If w_i is initially enabled, then similarly to the base case, it satisfies the condition in line 5. Otherwise, since $M_0 \xrightarrow{w}$, it must have become enabled after firing w_j for some $0 < j < i$, meaning w_j adds tokens to ${}^{\bullet}w_i$ and/or removes tokens from ${}^{\circ}w_i$. By induction hypothesis, w_j satisfies the condition of line 5, and by line 7-8 we have that $w_j^{\boxplus}$ and ${}^{\boxminus}w_j$ are not in $S_{\leq}$ and $S_{\geq}$, respectively. Hence, for all $p \in P$ whenever $W_{\boxminus}(p, w_i) > M_0(p)$ we must have $p \notin S_{\leq}$ and whenever $I(p, w_i) \leq M_0(p)$ we must have $p \notin S_{\geq}$, which means w_i satisfies the condition on line 5.

Algorithm 1: Rule M: Effectively dead places and transitions

Input: A net $N = \langle P, T, W_{\boxminus}, W_{\boxplus}, I \rangle$, initial marking M_0 and CTL* formula φ
Output: A reduced net N' and its initial marking M_0'

1 $S_{\leq} := P$ /* Places that definitely cannot gain tokens */
2 $S_{\geq} := P$ /* Places that definitely cannot lose tokens */
3 $F := T$ /* Transitions that definitely cannot fire */
4 **do**
  /* Find transitions that may fire and update sets accordingly */
5 **foreach** $t \in F$ *where* $\forall p \in P.(W_{\boxminus}(p,t) \leq M_0(p) \vee p \notin S_{\leq}) \wedge (I(p,t) > M_0(p) \vee p \notin S_{\geq})$ **do**
6 $F := F \setminus \{t\}$
7 $S_{\leq} := S_{\leq} \setminus t^{\boxplus}$
8 $S_{\geq} := S_{\geq} \setminus {}^{\boxminus}t$
9 **until** $S_{\leq}$, $S_{\geq}$, *and* F *do not change*
10 $P' := P \setminus (S_{\leq} \cap S_{\geq} \setminus \mathit{places}(\varphi))$
11 $T' := T \setminus F$
12 **return** $N' = \langle P', T', W_{\boxminus}, W_{\boxplus}, I \rangle$ *and* M_0

We can now conclude that all transitions in w satisfy the condition of line 5. Since w is the shortest trace such that $M_0 \xrightarrow{w} M$, the last transition must have been the transition that added or removed tokens from p such that $M_0(p) \neq M(p)$. However, then by line 7-8, we have that $p \notin S_{\leq} \cap S_{\geq}$, and then we have a contradiction, because we assumed p was removed, but only places in $S_{\leq} \cap S_{\geq}$ are removed.

Given a place p that never gains or loses tokens, it should be clear why any transition in $p^{\bullet} \cup p^{\circ}$ that is initially disabled/inhibited by p will stay disabled/inhibited, and any transition in $p^{\bullet} \cup p^{\circ}$ not disabled/inhibited will never be disabled/inhibited by p. Hence, unless $p \in \mathit{places}(\varphi)$, then p can be removed, as well as any transitions initially disabled/inhibited by p.

Furthermore, since all transitions in w satisfies the condition of line 5 and therefore removed from F in line 6, we can use a very similar approach to show that all transitions in F cannot fire. □

The Rule M is inspired by Rule 10 by Yann Thierry-Mieg in [21]. Rule 10 removes maximal unmarked siphons, which are places with zero tokens initially and which cannot gain more tokens, since their pre sets require tokens in places that are also in the unmarked siphon. Our Rule M is more general as the removed places do not have to be empty in the initial marking and they just have to be effectively dead. That is, they may have fireable transitions in their preset, but those transitions do not affect any removed place. Rule M is also more general than Rule E in TAPAAL [5]. Rule E finds a place that cannot gain tokens according to its local transitions and then removes transitions that require

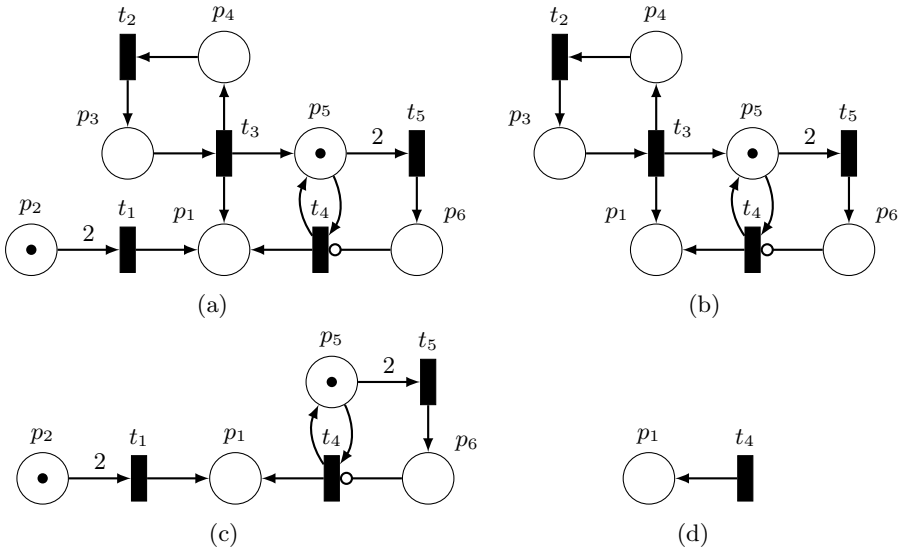


Figure 12: Example showing that Rule M supersedes Rule E [5] and Rule 10 [21]. (a) shows a Petri net N. (b) shows N with Rule E applied. (c) shows N with Rule 10 applied. (d) shows N with Rule M applied.

tokens to be added to the place to be enabled. Rule M supersedes Rule E as Rule M can determine non-increasing places from more context than just the local transitions.

Theorem 8. *Rule M in Algorithm 1 supersedes Rule E from [5] and Rule 10 from [21].*

Proof. The proof follows naturally from the definitions of what the respective rules remove. Figure 12 shows an example of Rule M reducing a net further than Rule 10 and Rule E. □

3.3.3 Rule Q: Preemptive transition firing Rule Q evaluates transitions that are initially enabled and are the only consumer of all places in its pre set. The formal description of Rule Q can be found in Figure 14. Rule Q is an extension of Rule 17 described by Thierry-Mieg in [21] and adds support for inhibitor arcs and allows the given transition to have more than one place in its pre set.

Remark 2. Rule Q can potentially put tokens into places which will prevent other reductions. Futhermore, it can be applied infinitely if $W_{\boxminus}(t_0) \leq W_{\boxplus}(t_0)$ or if the Petri net contains a loop like shown in Figure 13 where none of the places in the loop is in $places(\varphi)$ or has an inhibitor arc. However, Rule I [5] removes such problematic loops and Rule Q is therefore safe to use after an application of Rule I.

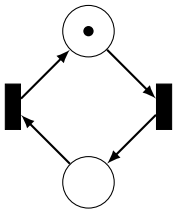
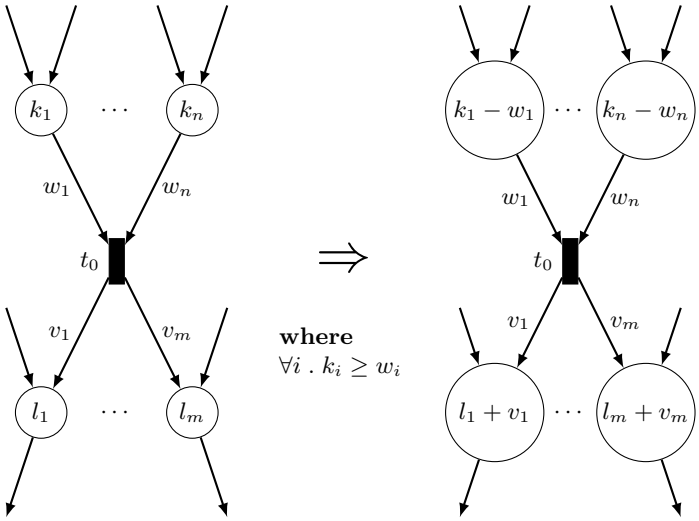


Figure 13: A loop where Rule Q can be applied infinitely.

As Thierry-Mieg notes, the Rule Q makes more post agglomerations possible since it empties potentially agglomerable places. We also see this in TAPAAL [5], where Rule A and B are special cases of post agglomerations where the agglomerated place is emptied as part of the rules. Hence, Rule Q generalises all such preemptive transition firing.



Precondition	Update
Fix transition t_0 s.t.: Q1) $({}^\bullet t)^\bullet = \{t_0\}$ Q2) $W_{\boxminus}(t_0) \leq M_0 < I(t_0)$ Q3) $({}^\bullet t_0 \cup t_0^\bullet) \cap places(\varphi) = \emptyset$ Q4) $({}^\bullet t_0)^\circ = (t_0^\bullet)^\circ = \emptyset$	UQ1) $M_0 := M_0 + E(t_0)$.

Figure 14: Rule Q: Preemptive transition firing

Theorem 9. *Rule Q in Figure 14 is correct for CTL*\X.*

Proof. Let $N = \langle P, T, W_{\boxminus}, W_{\boxplus}, I\rangle$ be a Petri net and let $M_0 \in \mathcal{M}(N)$ be its initial marking. Let $N' = \langle P', T', W'_{\boxminus}, W'_{\boxplus}, I'\rangle$ with initial marking $M'_0 \in \mathcal{M}(N')$ be the Petri net obtained by applying Rule Q on N once for an arbitrary formula $\varphi \in$ CTL*\X.

We argue that Rule Q is correct using an equivalence relation $\equiv_Q \subseteq \mathcal{M}(N) \times \mathcal{M}(N')$ such that $M \equiv_Q M'$ iff $M = M'$ or $M + E(t_0) = M'$.

Note that by Q2 we have that t_0 is fireable in the initial marking and by UQ1 we have that $M'_0 = M_0 + E(t_t)$, so $M_0 \equiv_Q M'_0$. Our proof then follows from the next three properties. Let $M \equiv_Q M'$, then

P1) M and M' satisfies the same atomic propositions w.r.t. φ, and
P2) if $M \xrightarrow{t} M_1$ then either $M_1 \equiv_Q M'$ or $M' \xrightarrow{t} M'_1$ such that $M_1 \equiv_Q M'_1$, and
P3) if $M' \xrightarrow{t} M'_1$ then either $M \xrightarrow{t} M_1$ or $M \xrightarrow{t_0 t} M_1$ such that $M'_1 \equiv M_1$.

We now argue for the three properties. By Q3, t_0 is invisible to φ and since $M \equiv_Q M'$ iff $M = M'$ or $M + E(t_0) = M'$, we have P1 is true.

For P2 there are two cases:

- Case $t = t_0$: We want to show that $M_1 \equiv M'$. By definition of $\equiv_Q$ and $M \xrightarrow{t} M_1$, we have that $M_1 = M + E(t) = M + E(t_0)$ and thus $M_1 \equiv M'$.
- Case $t \neq t_0$: We want to show that $M' \xrightarrow{t} M'_1$ such that $M_1 \equiv M'_1$. Together Q1 and $M \equiv_Q M'$ implies that $M(p) \leq M'(p)$ for all places $p \notin {}^\bullet t_0$ and thus by Q4 we know that t is enabled in M' so we can fire it such that $M' \xrightarrow{t} M'_1$. The transition t has the same effect in both nets, therefore $M - M' = M_1 - M'_1$ which implies that $M_1 \equiv_Q M'_1$.

There are two cases for P3 as well:

- Case $M = M'$: We want to show that $M \xrightarrow{t} M_1$ and $M_1 \equiv M'_1$. Clearly, since $M = M'$ and $M' \xrightarrow{t}$ we have that t is enabled in M and that $M \xrightarrow{t} M_1$ and $M_1 = M'_1$ implying $M_1 \equiv_Q M'_1$.
- Case $M + E(t_0) = M'$: We want to show that $M \xrightarrow{t_0 t} M_1$ and $M_1 \equiv M'_1$. By Q2, t_0 is initially enabled, and by Q1 and $M + E(t_0) = M'$ we know that t_0 is still enabled in M. Therefore, we can fire t_0 such that $M \xrightarrow{t_0} M_2$ and $M_2 = M'$. Since $M' \xrightarrow{t} M'_1$ we also have that $M_2 \xrightarrow{t} M_1$, and $M_1 = M'_1$ implying $M_1 \equiv_Q M'_1$.

By property P1, P2, and P3 the relation $\equiv_Q$ is a stuttering bisimulation, and by Theorem 1 we have that Rule Q is correct for any formula $\varphi \in$ CTL*\X. □

3.3.4 Rule R: Atomic post-agglomerable producer Rule R is similar to a post agglomeration rule [11] and a formal description of Rule R is in Figure 15. In a post agglomeration, a place is found where any tokens that enter can always be immediately forwarded without consequences. For every pairing of a

producer and a consumer of such a place, a new transition is made combining the two transitions. Lastly, the original place and its producers and consumers are removed. An agglomeration rule will not always reduce the size of the Petri net, but will always reduce the number of interleavings and thereby the size of the state space. This follows from the fact that tokens can no longer stay at the agglomerated place.

The Rule R which we present now takes the post agglomeration a bit further. In Rule R we instead look for a place p_0 with a producer t_0 such that t_0 can always be followed by a firing of any consumer of p_0 without inhibiting other transitions or affecting places in $places(\varphi)$. The producer t_0 is then replaced with new transitions, one for each consumer, and these new transitions combine the effect of firing t_0 and the given consumer. Similarly to an agglomeration rule, Rule R removes interleavings despite potentially increasing the size of the Petri net. However, Rule R is more general, since it only operates on one producer at a time and leaves p_0 untouched, allowing tokens in p_0 in the initial marking, which a post agglomeration does not. Additionally, Rule R does not require the weights of the arcs to and from the agglomerated place to be equal, making R usable in many cases.

Theorem 10. *Rule R in Figure 15 is correct for LTL\X.*

Proof. Let $N = \langle P, T, W_{\boxminus}, W_{\boxplus}, I\rangle$ be a Petri net and let $M_0 \in \mathcal{M}(N)$ be its initial marking. Let $N' = \langle P', T', W'_{\boxminus}, W'_{\boxplus}, I'\rangle$ with initial marking $M'_0 \in \mathcal{M}(N')$ be the Petri net obtained by applying Rule R on N once for an arbitrary formula $\varphi \in$ LTL\X. Since Rule R does not modify places, the initial markings M_0 and M'_0 are the same, but we use the prime to indicate that we are using the reduced net.

To argue for the correctness of Rule R, we argue that any firing sequence (finite or infinite) of $\langle N, M_0\rangle$ has a stutter equivalent firing sequence in $\langle N', M'_0\rangle$ and vice versa.

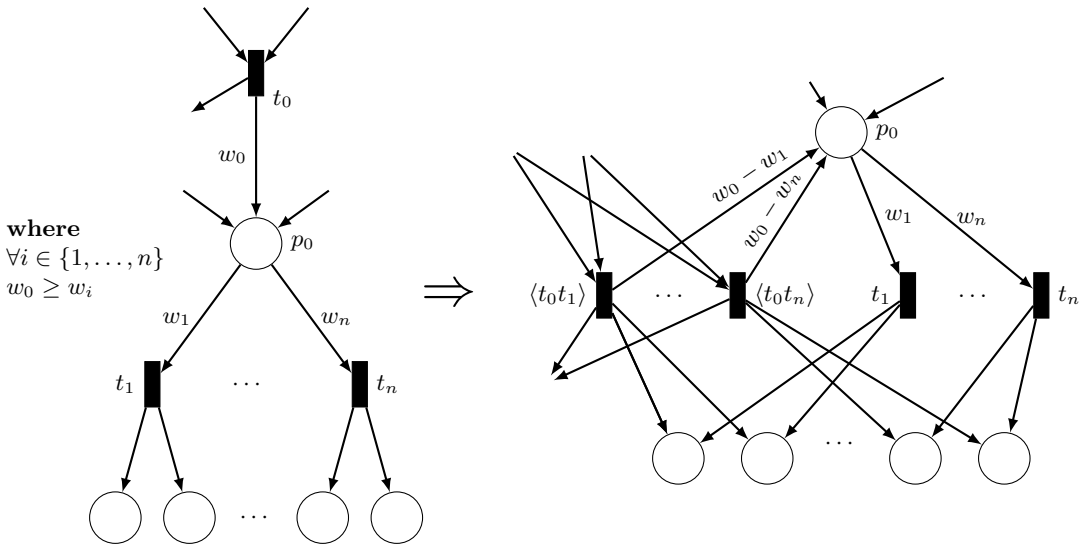
$\Rightarrow$: Let $M_0 \xrightarrow{w}$. We will now show that $M'_0 \xrightarrow{\mathcal{T}(w)}$ where $\mathcal{T}(w)$ is a stutter equivalent translation of w defined inductively as:

$$\mathcal{T}(u) = u \tag{1}$$
$$\mathcal{T}(vt_0x) = \mathcal{T}(v\langle t_0t\rangle x) \qquad \text{for an arbitrary } t \in p_0^\bullet \tag{2}$$
$$\mathcal{T}(vt_0ytz) = \mathcal{T}(v\langle t_0t\rangle yz) \qquad \text{for all } t \in p_0^\bullet \tag{3}$$

where $u \in T'^{\bigstar}$, $v \in T'^{*}$, $x \in (T \setminus p_0^\bullet)^{\bigstar}$, $y \in (T \setminus p_0^\bullet)^{*}$, $z \in T^{\bigstar}$. The transition $\langle t_0t\rangle$ is the added transition equal to the firing sequence t_0t for some $t \in p_0^\bullet$ as by UR1. The translation function $\mathcal{T}$ removes firings of t_0 by either inserting or moving forward the firing of a transition from $p_0^\bullet$, and then combining the two.

The case (1) is correct, because the removed transition t_0 does not appear in u, and hence, the reduced net N' can match all the transitions made by the original net N. Clearly, they are stutter equivalent too.

The case (2) is correct, because a $t \in p_0^\bullet$ exists by R1, and by R3, R5, and R6 we know that t is enabled after one firing of t_0. By R5 and since $x \in (T \setminus p_0^\bullet)^{\bigstar}$



Precondition	Update
Fix place p_0 and transition t_0 s.t.: R1) $t_0 \in {}^\bullet p_0 \wedge p_0^\bullet \neq \emptyset$ R2) ${}^\bullet p_0 \cap p_0^\bullet = \emptyset$ R3) $p_0^\circ = {}^\circ(p_0^\bullet) = (({p_0}^\bullet)^\bullet)^\circ = \emptyset$ R4) $(\{p_0\} \cup (p_0^\bullet)^\bullet) \cap places(\varphi) = \emptyset$ R5) ${}^\bullet(p_0^\bullet) = \{p_0\}$ R6) $W_\boxplus(p_0, t_0) \geq W_\boxminus(p_0, t)$ for all $t \in p_0^\bullet$	UR1) For each transition $t \in p_0^\bullet$ create a transition $\langle t_0 t \rangle$ with the following arcs: $W_\boxminus(\langle t_0 t \rangle) = W_\boxminus(t_0)$ $W_\boxplus(\langle t_0 t \rangle) = W_\boxplus(t_0) + W_\boxplus(t) - W_\boxminus(t)$ $I(\langle t_0 t \rangle) = I(t_0)$ UR2) Remove t_0

Figure 15: Rule R: Atomic post-agglomerable producer

the tokens added to p_0 by t_0 remains unused throughout the firing of x. Hence, we can fire some $t \in p_0^\bullet$ right after firing t_0, and $\langle t_0 t \rangle$ has the same effect as the sequence $t_0 t$ as by UR1. Additionally, by R4 t is invisible to φ and thus does not affect the satisfaction of φ. This means t creates a stutter and that $v\langle t_0 t \rangle x$ is stutter equivalent to $v t_0 x$.

The case (3) is correct, because by R3, R5, and R6 we know that t is enabled after one firing of t_0. By R3 and since $y \in (T \setminus p_0^\bullet)^*$, we have that t stays enabled during the firing of y. Therefore, we can fire t right after t_0, and $\langle t_0 t \rangle$ has the same effect as the sequence $t_0 t$ as by UR1. Additionally, by R4 t is invisible to φ and thus $v\langle t_0 t \rangle yz$ is stutter equivalent to $v t_0 y t z$.

Hence, for all $w \in T^\bigstar$ such that $M_0 \xrightarrow{w}$ we have a fireable and stutter equivalent $\mathcal{T}(w)$ such that $M_0' \xrightarrow{\mathcal{T}(w)}$.

$\Leftarrow$: Let $M_0' \xrightarrow{w}$. We will now show that $M_0 \xrightarrow{\mathcal{T}'(w)}$ where $\mathcal{T}'(w)$ is a stutter equivalent translation of w defined inductively as:

$$\mathcal{T}'(u) = u \tag{4}$$
$$\mathcal{T}'(v\langle t_0 t\rangle x) = \mathcal{T}'(v t_0 t x) \tag{5}$$

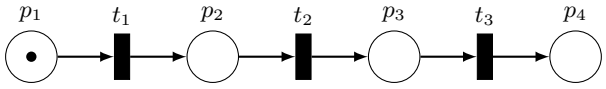
where $u \in T^{\bigstar}$, $v \in T^*$, $x \in T'^{\bigstar}$, and where and $\langle t_0 t\rangle$ for any $t \in p_0^\bullet$ is one the constructed transitions equal to firing sequence $t_0 t$ as by UR1. The translation function $\mathcal{T}'$ replaces the transition $\langle t_0 t\rangle$ with the equivalent sequence $t_0 t$ where $t \in p^\bullet$.

The case (4) is correct, because the sequence u only contains transitions which are present in the original net N and therefore the original net N can match every transition. Clearly, they are stutter equivalent.

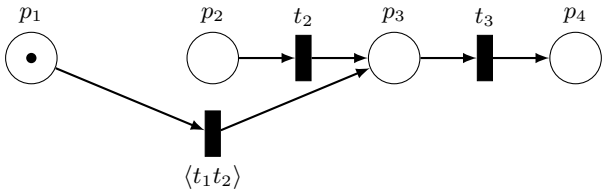
The case (5) is correct, because the added transition $\langle t_0 t\rangle$ has the same effect as the sequence $t_0 t$ as by UR1. By R4 t is invisible to φ and thus $v\langle t_0 t\rangle x$ is stutter equivalent to $v t_0 t x$.

We have now shown that for all firing sequences in one net, the other net has a fireable and stutter equivalent firing sequence. Hence, by Theorem 2, for any formula $\varphi \in \text{LTL}\backslash\text{X}$ we have $M_0 \vDash \varphi$ iff $M_0' \vDash \varphi$. □

The Rule R is stutter insensitive since it reduces the length of firing sequences. That means properties that involve counting (such as the temporal next operator X) are not preserved. Figure 16 shows a counter-example of why Rule R does not work for LTL. The Rule R is not applicable for CTL\X properties either since it removes branching options. Figure 17b shows an example of this.

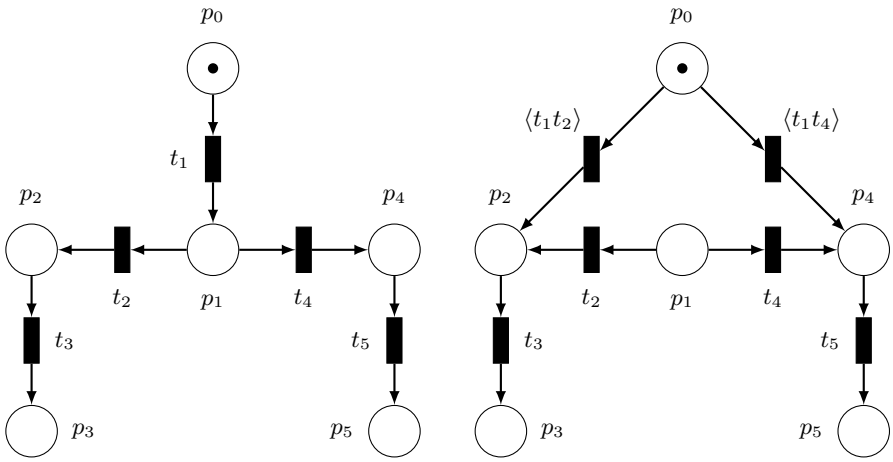


(a) Original net N where we have $N \vDash XXp_4 = 0$



(b) Reduced net N' where we have $N' \nvDash XXp_4 = 0$

Figure 16: Counter-example showing why Rule R does not preserve stutter-sensitive properties. (a) shows Petri net N and (b) shows the reduced Petri net N' which is N after one application of Rule R w.r.t the LTL property $XXp_4 = 0$ (The number of tokens at p_4 must be 0 after two steps). However, the property is not preserved.



(a) Petri net N where $N \vDash EF(p_0 = 0 \wedge (EF\ p_3 = 1) \wedge (EF\ p_5 = 1))$

(b) Reduced Petri net N' where $N' \nvDash EF(p_0 = 0 \wedge (EF\ p_3 = 1) \wedge (EF\ p_5 = 1))$

Figure 17: Counter-example showing why Rule R does not work for CTL\X properties. (a) shows Petri net N. (b) shows Petri net N' which is N after one application of Rule R on (p_1, t_1) w.r.t to the CTL\X property $EF(p_0 = 0 \wedge (EF\ p_3 = 1) \wedge (EF\ p_5 = 1))$ stating that the net can reach a marking where p_0 is empty, and from there it can reach a marking where p_3 has a token and it can also reach a marking where p_5 has a token. However, the property is not preserved.

We argue that Rule R is more general than a traditional post agglomeration [10], except that it does not remove the agglomerated place, but this can be achieved by applying a clean-up rule afterwards, e.g. Rule E or M. Figure 18 shows a post-agglomerable net and how Rule R followed by Rule E reaches what a traditional post agglomeration achieves in one step. Note, however, that if there are tokens in place p_0, then a traditional post agglomeration cannot be performed, unless Rule Q empties it first. But Rule Q requires that there is only one way the tokens can be consumed, i.e. one consumer, and that the number of tokens is divisible by the weight to this consumer. Rule R on the other hand can still be used under those conditions to build the interleavings, so the use of Rule E to remove the place afterwards is the only part of the reduction that is unavailable.

Theorem 11. *The rule application sequence R*E* supersedess traditional post agglomerations described in [10].*

Proof. As argued above. Figure 18 shows a post-agglomerable net and how exhaustive application of Rule R followed by Rule E reaches what a traditional post agglomeration achieves in one step. □

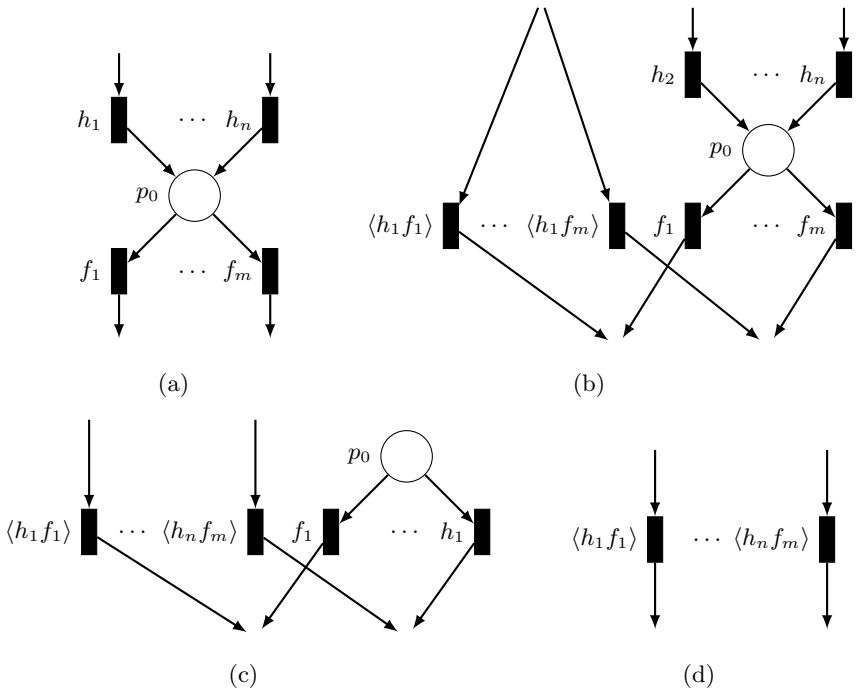


Figure 18: Example of using Rule R and Rule E in combination to achieve a traditional post agglomeration. (a) shows a post-agglomerable place of Petri net N. (b) shows N with R applied. (c) shows N with R applied exhaustively. (d) shows N with R applied exhaustively and then E applied exhaustively.

The real strength of Rule R is that it can also handle weights—even if the weight is different on the producer-place arc and the place-consumer arcs. Exhaustive application of Rule R builds all permutations of fire orderings of the consumers that consume less than the number of tokens added to the place by the producer. For instance, if producer t_0 adds 4 tokens to post-agglomerable place p_0 with two consumers t_1 and t_2, consuming 1 and 2 tokens respectively. Then, after applying Rule R exhaustively, we end up with the interleavings: $\langle t_0 t_1 t_1 t_1 t_1 \rangle, \langle t_0 t_1 t_1 t_2 \rangle, \langle t_0 t_1 t_2 t_1 \rangle, \langle t_0 t_2 t_1 t_1 \rangle$, and $\langle t_0 t_2 t_2 \rangle$. In this case there are no left-over tokens, but if there are, those would be added to p_0. A concern is that some interleavings have the same effect. Rule R builds every firing permutation, while ideally we only need every combination. Rule C [5] or Rule L can remove these excess interleavings.

Due to Rule R being able to handle weights, it can in combination with Rule Q and Rule E also achieve the same reductions as both Rule A and Rule B of [5]. Figure 19 shows an example of Rule R and Rule E achieving the same reduction as Rule A. If there are tokens in p_0 then Rule Q is also be needed.

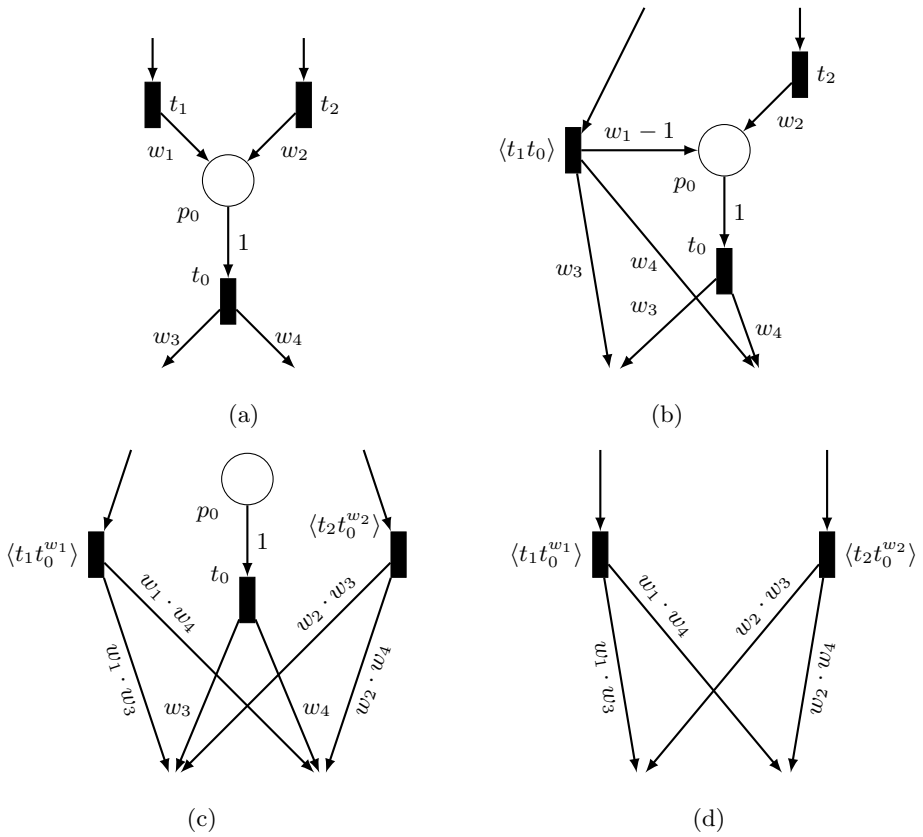


Figure 19: Example of Rule R and Rule E performing a reduction equivalent to Rule A [5]. (a) shows a Petri net N. (b) shows N where Rule R has been applied once. (c) shows N where Rule R has been applied exhaustively. (d) shows N where Rule R and E has been applied exhaustively.

Theorem 12. *The rule application sequence QR*E supersedess both Rule A and Rule B from [5].*

Proof. As argued above. Figure 19 shows an example of Rule R and Rule E achieving the same reduction as Rule A. This example can be easily generalized. For Rule B a similar approach is be used and it is not hard to see. □

Remark 3. The implementation of Rule B in TAPAAL differs from the specification given in [5] and captures a similar scenario, where the given place is pre agglomerable instead.

4 Experiments and results

We implement the described fixes, extensions, and new rules from Section 3 in `verifypn` [13, 4] which is the verification engine used in the Petri net verification tool TAPAAL [8]. The engine `verifypn` is written in C++ and our modifications are mainly to `Reducer.cpp`. Our modified version of `verifypn` can be found on GitHub.

We experimentally test the performance of our modifications fixes, extensions, and new rules using the Petri net models from the MCC2021 [16] competition. The set of models consists of 1181 Petri net models of varying sizes. We use the category ReachabilityCardinality and the first 8 properties for each model. This gives us a test suite of 9448 queries. Some experiments use a modified MCC2021 dataset where a random 10% of in-arcs have been replaced with inhibitor arcs. This creates non-sensible models and queries, but we do this to check the impact of our rules that extend existing rules with support for inhibitor arcs. The hardware we use for testing is the DEIS MCC, which is a compute cluster at Aalborg University, configured for repeatability and stability and memory-intensive computations. Each query is allocated a maximum of 15GB memory and 2 cores on a Xeon e5 2680 CPU. If the memory limit is reached, the query is automatically terminated.

In Section 4.1 we introduce the specific experiments that we run, the notation we use for experiments and the order in which they apply the rules, as well as the metrics that we measure. In Section 4.2 we study if our proposed fixes presented in Section 3.1 to TAPAAL increases the performance. In Section 4.3 we look at the effect of our proposed extensions and new rules presented in Section 3.2 and 3.3, respectively. We then present results from testing on the MCC2021 test suite with inhibitor arcs. These results are presented in Section 4.4. Finally, we study the applicability of rules in Section 4.5.

4.1 Experiments

Experiments apply a sequence of structural reductions repeatedly until exhaustion, i.e no more rules are applicable, or until a timeout is reached. We use regex syntax to easier define these sequences of rules. An example of this is the sequence of rules A*D*C*, where the experiment tries to apply Rule A first as many times as possible, then D as many times as possible, then C. We denote the rule sequence A*B*C*D*E*F*G*I* as base since these are the rules already existing in `verifypn`.

When testing a new rule X, we append the rule X to the base rule sequence, i.e (base.X*)*, and we simply refer to these experiments by their rule sequence.

We also run an experiment TAPAAL, which uses the rule sequence (base)*, except that it uses the original implementation of E and I without our fixes. We can then evaluate how big of an impact the fixes have. We also run an experiment that appends all of our new rules and rule extensions to the base sequence, (base.L*M*N*O*P*Q*R)*, as well as (A*B*C*D*M*F*G*H*I*N*O*P*)* with

our most effective rules. Note that Rule N and O shares implementation and is therefore never used separately.

In these experiments, we measure multiple metrics. The reduce time is time spent applying the reduction rules in the rule sequence given by the experiment name. The verification time is the time spent verifying the property in the reduced net. Verification is done using query simplification and the Certain Zero algorithm [7]. The total time is the sum of reduce time and verification time. We also measure the state space size of the reduced net, as this allows us to compare how effective the rules are at reducing the size of the state space. Another metric is rule application. This is the percentage of models where a rule was applied at least once while reducing w.r.t to any of the 8 properties. We also count the number of answers each experiment get, out of the 9448 possible queries. The structural reduction phase, the verification phase and time spent exploring the state space each have a timeout of 1 minute. If the entire state space is not explored within 1 minute, the size is not reported.

4.2 Fixes to TAPAAL results

In this section, we compare results from the experiments TAPAAL and (base)*. The experiments use the same rule sequence, but (base)* includes our fixes to Rule E and I.

In Figure 20a, 20c, and 20d, we respectively see the total times, reduce times, and verification times of the two experiments. Only results above 10 seconds are shown. We see that the results from the experiments TAPAAL and (base)* are very similar, however with (base)* having a slight improvement, especially visible on Figure 20c showing the time spent in the reduction phase. It should be noted that these small differences in the results, can be due to noise.

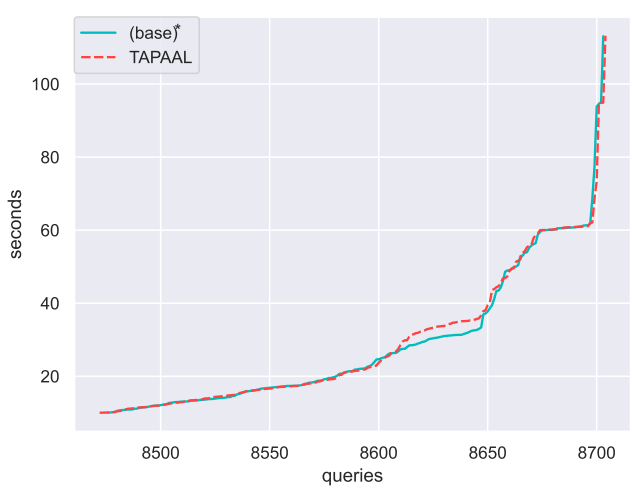
In Figure 20b we show the 5% biggest state space sizes explored within the timeout for TAPAAL and (base)*. Here we see that the results are almost identical, but that (base)* managed to explore the state space for a few additional models. Again, the difference is likely due to noise.

TAPAAL and (base)* agree on all found answers in this test suite, despite the fact that the implementation of TAPAAL does not correspond to the specification as discussed in Section 3.1. In the following sections, we do not compare experiments to TAPAAL since (base)* is so similar.

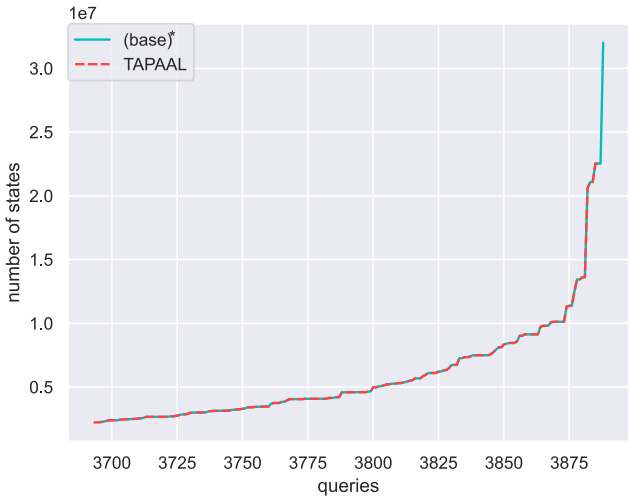
4.3 Non-inhibitor test suite results

In this section we look at the results for our new rules and rule extensions when running on MCC2021 reachability cardinality queries. The total times are shown in Figure 22a. We see that the rule sequences (base.N*O*)* and (A*B*C*D*M*F*G*H*I*N*O*)* slightly outperforms (base)*, while (base.M*)* and (base.Q*)* are similar to (base)*. The rule sequence (base.L*)* is slower than (base)* for queries below 60 seconds, but catches up after that.

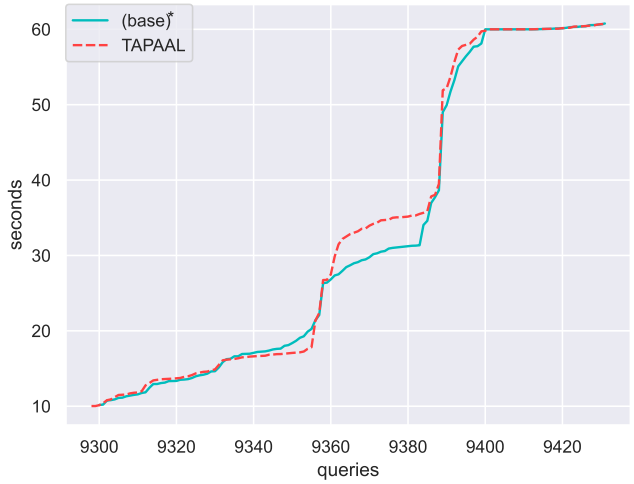
In Figure 22b the sizes of the reduced state space is shown. Appending any of our rules to the base rule sequence results in further reduced state spaces. It



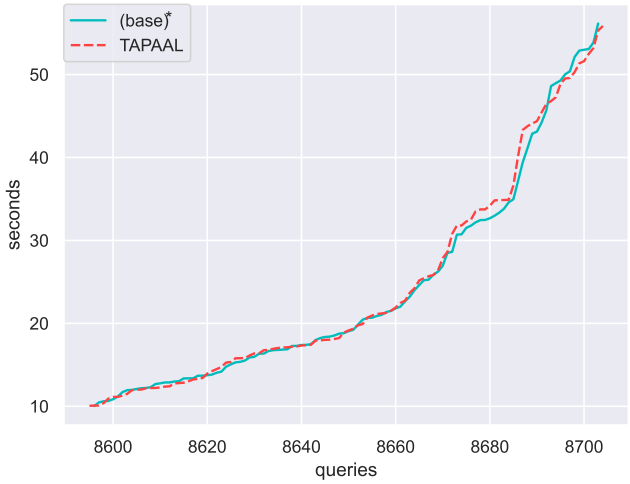
(a) Total time



(b) State space size

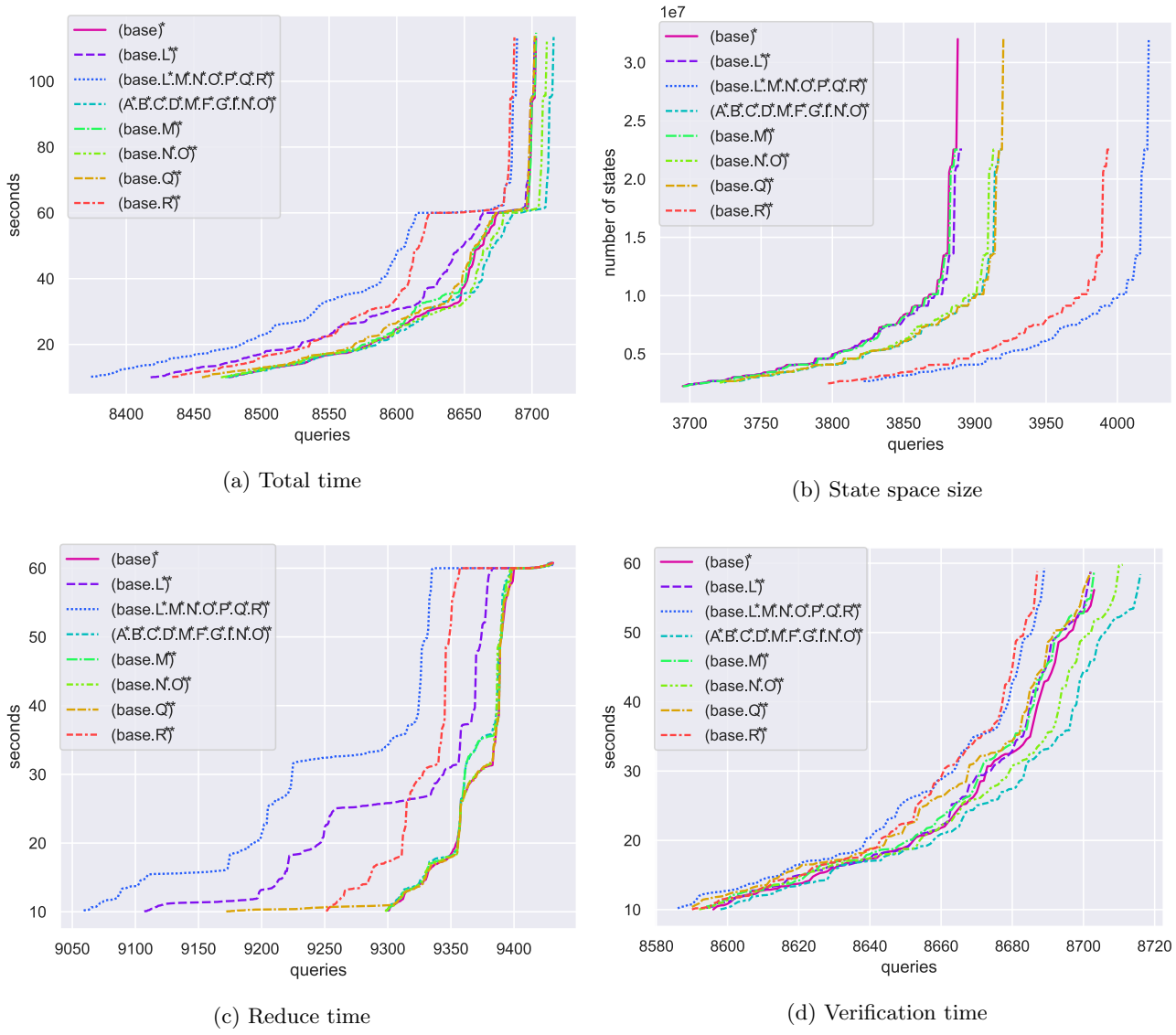


(c) Reduce time



(d) Verification time

Figure 21: Comparing TAPAAL experiment to (base)* which includes our fixes to rules E and I. 9448 queries in total. (a) Total time spent in reduction and verification phase for each experiment. (b) Top 5% state space sizes of reduced nets explored within timeout for each experiment. (c) Time spent in reduction phase for each experiment. (d) Time spent in verification phase for each experiment. (a),(c), and (d) show results above 10 seconds, and all data points are sorted by size or time.



(a) Total time

(b) State space size

(c) Reduce time

(d) Verification time

Figure 23: Non-inhibitor test suite comparing new rules to (base)* on test suite of 9448 queries. (a) Total time spent in reduction and verification phase for each experiment. (b) Top 5% state space sizes found within timeout of reduced nets for each experiment. (c) Time spent in reduction phase for each experiment. (d) Time spent in verification phase for each experiment. (a),(c), and (d) we show results above 10 seconds, and all data points are sorted by size.

is Rules-NO, Q and particularly R that enables further state space reduction. It is counter-intuitive that Rule R reduces the state space size so significantly but has a negative impact on the verification time. We discuss this in Section 5.1.

We decompose total time into reduce time and verification time in Figure 22c and Figure 22d, respectively. Rules NO have a small positive impact on the verification time while Rule R have a small negative impact. The rest of the rules have almost no impact. However, in (A*B*C*D*M*F*G*I*N*O)* where Rule M replaces E compared to (base.N*O*)*, we see that Rule M improves the verification time even further, which makes sense since Rule M supersedes E. We also see that this experiment is the one that gets the most answers within the time limit, specifically 13 more answered queries than that of (base)*. Unsurprisingly, the more reduction rules are used, the more overhead the reduce phase has. We see from (base.M*)*, (base.N*.O*)*, and (base.Q*)* that Rule M, NO, and Q have low or no overhead, while from (base.L*)* and (base.R*)* we see that Rule L and R have some overhead.

4.4 Inhibitor test suite results

To test the performance of our rules on Petri nets with inhibitor arcs, we use the modified MCC2021 test suite containing inhibitor arcs. The arc conversion makes the properties nonsensical, as we do not modify these to reflect the changes in the models. However, we still find whether or not we preserve the answer to the property when using our new rules, and we are still able to see if there is a performance improvement, as we have done with the normal test suite in Section 4.3. We run fewer experiments on this test suite as we are mainly interested in our extension rules P and N that specifically concerns inhibitor arcs.

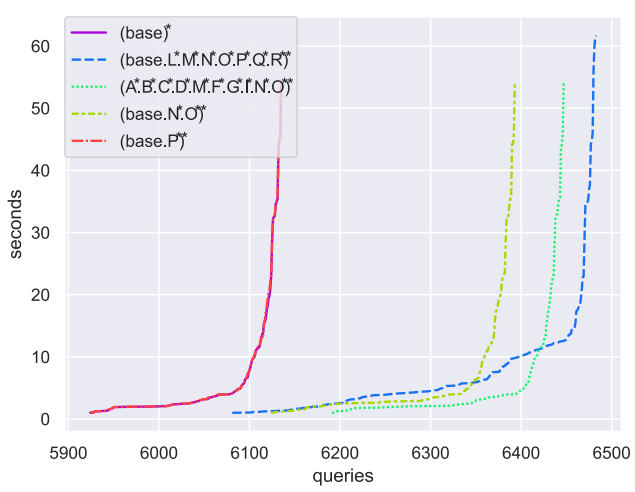
In Figure 24a we see the summed total time above 1 second for experiments on the inhibitor data suite. We see that (base)* finds 6152 answers within the time limits, while (base.N*O*)* finds 6410 answers (4.19% more) and (base.L*M*N*O*P*Q*R*)* finds 6500 answers (5.65% more). The experiment (base.P*)* shows no improvements over (base)*.

In Figure 24b we see the sorted state space sizes found within the timeout. Not surprisingly, using all rules results in the smallest state spaces out of our experiments. The experiment (base.N*O*)* is significantly better than (base)*, while (base.P*)* barely is. Note that with all rules enabled, we are able to search through 281 more state spaces than (base)* within the timeout.

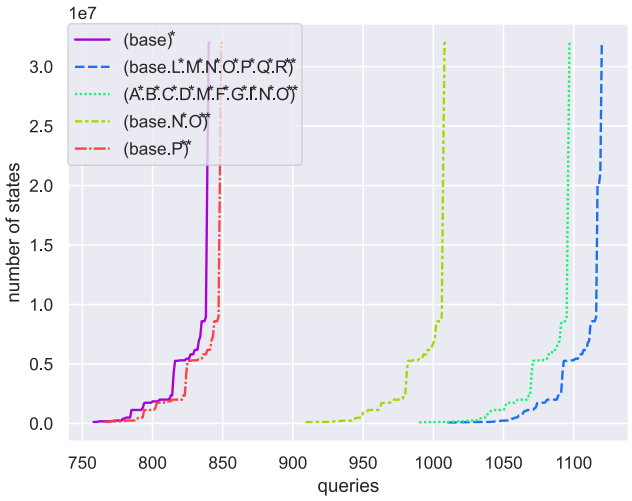
Figure 24c shows the sorted reduce times above 10 seconds, and Figure 24d shows the sorted verification times above 1 second. In this test suite, the reduce time overhead from our extensions and new rules is negligible compared to the improvements we see in the verification time. This is clearly shown from the number of answers experiments using the new rules are able to get.

4.5 Rule application

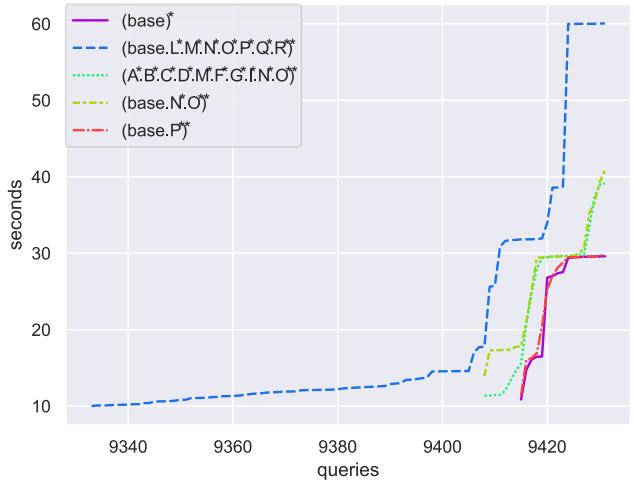
Table 1 shows the percentage of models where each rule was used in at least one query. These percentages show how often a rule is applicable. There are a few



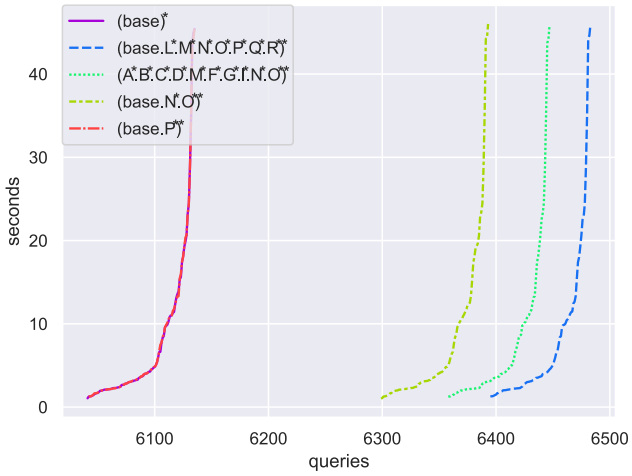
(a) Total time



(b) State space size



(c) Reduce time



(d) Verification time

Figure 25: Comparing rules on the inhibitor test suite of 9448 queries. (a) Total time spent in reduction and verification phase for each experiment. (b) Top 10% state space sizes of reduced nets found within timeout for each experiment. (c) Time spent in reduction phase for each experiment. (d) Time spent in verification phase for each experiment. (a) and (d) show results above 1 seconds, (c) shows results above 10 seconds, and all data points are sorted.

Row	Experiment \ Rule	A	B	C	D	E	F	G	H	I	L	M	NO	P	Q	R
1	**Non-inhibitor test suite**															
2	TAPAAL	77.6	51.2	29.1	41.6	4.9	59.1	44.2	23.0	41.3	-	-	-	-	-	-
3	(base)*	77.6	51.2	29.1	41.6	5.3	59.2	44.2	23.0	41.3	-	-	-	-	-	-
4	(base.L*)*	77.6	51.2	29.1	41.6	5.4	59.2	44.2	23.0	41.3	6.4	-	-	-	-	-
5	(base.M*)*	77.6	51.2	29.1	41.5	5.2	58.7	44.2	23.0	41.2	-	7.9	-	-	-	-
6	(base.N*.O*)*	78.1	51.3	29.1	42.2	5.3	59.8	44.2	23.0	41.3	-	-	10.2	-	-	-
7	(base.Q*)*	78.2	51.9	29.7	42.1	12.0	63.5	44.2	23.4	41.8	-	-	-	-	26.9	-
8	(base.R*)*	77.4	51.2	29.5	42.4	5.4	60.5	46.5	24.0	43.1	-	-	-	-	-	52.4
9	(base.L*.M*.N*.O*.P*.Q*.R*)*	76.6	52.0	30.0	42.9	13.0	64.4	46.6	23.6	43.2	6.4	8.1	3.3	0.0	27.1	53.0
10	(A*.B*.C*.D*.M*.F*.G*.I*.N*.O*)*	78.1	51.3	29.1	42.3	-	24.5	44.2	23.0	41.2	-	46.1	10.2	-	-	-
11	(M*.E*)*	-	-	-	-	0.0	-	-	-	-	-	11.5	-	-	-	-
12	((Q*.R*.E*)*.(A*.B*))*	0.0	40.2	-	-	40.0	-	-	-	-	-	-	-	-	59.5	79.5
12	**Inhibitor test suite**															
13	(base)*	67.3	37.3	22.4	35.1	4.4	38.0	38.0	14.7	73.8	-	-	-	-	-	-
14	(base.P*)*	67.3	37.2	22.3	35.1	53.5	38.1	38.0	14.7	73.8	-	-	-	53.0	-	-
15	(base.N*.O*)*	68.7	37.9	27.7	40.3	25.0	54.6	41.3	15.2	76.4	-	-	33.6	-	-	-
16	(A*.B*.C*.D*.M*.F*.G*.I*.N*.O*)*	68.1	37.8	23.0	40.5	-	41.1	41.6	15.1	79.0	-	74.4	9.9	-	-	-
17	(base.L*.M*.N*.O*.P*.Q*.R*)*	61.7	39.9	23.4	44.0	13.2	52.0	45.4	15.2	77.8	8.1	69.7	9.4	10.0	16.3	51.1

Table 1: Percentage of models with at least one application of the given rule in any query. A dash "-" indicates that the rule is not in the rule application sequence.

interesting things to note. For instance, as seen in Section 4.3, Rule L visibly impacts the reduce time, despite only being applicable in 6.4% of the models when using rule sequence (base.L*)* or (base.L*M*N*O*P*Q*R*)* as seen in Table 1, indicating that it is a costly rule to apply. Rule R also visibly impacts the reduce time, but is applicable in 52.4% of models when using (base.R*)* or 53.0% of the models when using (base.L*M*N*O*P*Q*R*)*. This is a high percentage considering that Rule A and Rule B are used earlier in the sequences and performs a similar reduction. Another thing to note is that the application of Rule F drops significantly when used right after Rule M in (A*B*C*D*M*F*G*H*I*N*O*)*. This is because Rule M can remove some of the lower-bounded places that Rule F also removes. The Rule M is inserted at E's place in the experiment (A*B*C*D*M*F*G*H*I*N*O*)* because we showed in Section 3.3.2 that M supersedes E and the experiment (M*E*)* confirms this. Interestingly, the applicability of Rule M is much higher in (A*B*C*D*M*F*G*H*I*N*O*)* than in other experiments, even in the experiment (M*E*)* where it is the first to be applied. We also try the rule sequence ((Q*R*E*)*(A*B*)*)* to show that Rules Q, R, and E supersedes Rule A and B. But as seen in the table, Rule B still finds application. The reason for this is that the implementation of Rule B is slightly different from the specification in [5] and makes it handle a similar pre-agglomeration reduction.

The applicability of Rule M and I in the inhibitor test suite are high, which can be explained by their ability to remove effectively dead parts and parts irrelevant to the query. As inhibitor arcs have been introduced randomly in the models, it is likely that some part of each model has become effectively dead or irrelevant. The applicability of the other base rules, A, B, C, D, E, F, G, and H, are slightly lower, than on the non-inhibitor test suite.

5 Discussion

In this section we discuss the unexpected results from testing Rule R, and then discuss our test suite and improvement on this.

5.1 Rule R discussion

Rule R is the rule that reduces the state space size the most according to the (base.R*)* experiment as seen in Figure 22b, but counter-intuitively it has a higher verification time than the (base)* experiment as seen in Figure 22d. This is not what we expect, as intuitively the smaller state space is, the faster it is to enumerate and find the goal state(s). The Rule R post-agglomerates on one producer at a time allowing it to remove many interleavings in the underlying state space. After multiple applications of Rule R the resulting Petri net has many transitions, each equal to one of these possible interleavings, and in certain nets, this will result in a huge branching factor in the underlying transition system. We believe this impacts the efficiency of the search strategy BestFS which uses a heuristic approximating the distance to a marking satisfying the atomic propositions of the property. The BestFS heuristic search is described in [14] by Finnemann et al. Since we have a huge branching factor, all the successors are added to a priority queue of the search at the same time, and the heuristic is run on each of these successors. The higher the branching factor is, the greater an impact the performance of the heuristic has. We have a hypothesis that using a depth-first search strategy in these nets will increase the performance compared to BestFS, at least for nets with high concurrency. We leave the investigation of this to future work.

5.1.1 Further testing As presented in Section 4 our test suite includes only half of the available properties in the reachability cardinality category. This allows us to run more experiments, test different implementations of rules, and different rule combinations. However, testing on the entire property set can show a more definitive answer on the effectiveness of the new rules, as there simply are more data points to compare.

Similarly, the MCC2021 dataset includes more categories than we use, e.g CTL/LTL fireability, CTL/LTL cardinality, and upper bounds. Testing on all these different logics and properties would allow us to better find bugs in the implementation, and some rules may be more effective for certain logics than others.

We use a timeout of one minute for each phase in our testing, but increasing the timeout can also show different results. In previous Section 5.1 we argue that using a DFS in combination with Rule R can increase its performance. We believe this would increase even more if done on larger models, where the goal states are hidden further into the state space, where the performance gap between BestFS and DFS might be larger.

In our test setup, we do not measure the memory used during testing, but we believe it is correlated with the size of the reduced models and the state space size. This is also an interesting metric to measure, as it is also part of the Model Checking Competition to have the lowest memory usage.

6 Conclusion

We presented further work on the structural reduction rules in the model checking tool TAPAAL described in [5]. We presented fixes to the existing rules Rule E and Rule I such that the implementation now agrees with the specification given in [5]. We presented three extensions to existing rules. Our Rule N and Rule O find a local lower bound on the number of tokens in a place and respectively use this to remove redundant arcs and always inhibited transitions. This extends on the existing Rule F which only uses the lower bound to remove the place if it never disables any transitions. Due to their similar behaviour Rule F, N, and O shares implementation. Our third rule extension is Rule P which removes redundant inhibitor arcs from places that cannot gain tokens. We also added two rules presented by Thierry-Mieg in [21] and extended them with support for inhibitor arcs. These are Rule L and Q. Furthermore, we presented two novel rules. Rule M finds and removes all effectively dead parts of the Petri net using fixed-point iteration, and Rule R is a more general post agglomeration that agglomerates on a single producer at a time. For each rule extension and each new rule, we proved its correctness for CTL* or a subclass of this logic.

Additionally, we prove that if Rule M is applied exhaustively, then Rule E cannot be applied. Similarly, we prove that if Rule Q and R are applied exhaustively, then Rule A and B cannot be applied. This indicates, that our new rules supersede the original A, B and E rules.

Lastly, we experimentally tested the presented rules on Model Checking Contest 2021 reachability cardinality queries in various experiments. We also convert some arcs to inhibitor arcs to test our support for those. We find that each of the presented rules combined with the rules presented in [5] allows for further reductions in the state space than the rules of [5] alone. Our rule fixes had a small positive impact. Rule L, M and N have the best impact on the number of queries answered within the verification time limit, however, Rule L has a large overhead considering its applicability. The Rule R has the biggest impact on the reduced net's state space size, but counter-intuitively, does not perform well during verification. We believe that it creates a huge branching factor, which affects the performance of the search strategy. This is yet to be verified. On the test suite with inhibitor arcs, our rules and extensions perform even better.

As future work, we would like to investigate structural reductions for coloured Petri net and investigate the results from Rule R as it shows good potential with state space reduction.

Acknowledgement We would like to thank Peter G. Jensen and Jiří Srba for supervising.

References

[1] Gérard Berthelot. "Checking properties of nets using transformation". In: *Applications and Theory in Petri Nets*. 1985.

[2] Gérard Berthelot, Gérard Roucairol and Rüdiger Valk. "Reductions of nets and parallel programs". In: *Net theory and applications*. Springer, 1980, pp. 277–290.

[3] M.C. Browne, E.M. Clarke and O. Grümberg. "Characterizing finite Kripke structures in propositional temporal logic". In: *Theoretical Computer Science* 59.1 (1988), pp. 115–131. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(88)90098-9. URL: https://www.sciencedirect.com/science/article/pii/0304397588900989.

[4] Frederik Bønneland et al. "Simplification of CTL Formulae for Efficient Model Checking of Petri Nets". In: Jan. 2018, pp. 143–163. ISBN: 978-3-319-91267-7. DOI: 10.1007/978-3-319-91268-4_8.

[5] Frederik M. Bønneland et al. "Stubborn versus structural reductions for Petri nets". In: *Journal of Logical and Algebraic Methods in Programming* 102 (2019), pp. 46–63. ISSN: 2352-2208. DOI: https://doi.org/10.1016/j.jlamp.2018.09.002. URL: https://www.sciencedirect.com/science/article/pii/S235222081830035X.

[6] Edmund M Clarke et al. *Handbook of model checking*. Vol. 10. Springer, 2018.

[7] Andreas E. Dalsgaard et al. "Extended Dependency Graphs and Efficient Distributed Fixed-Point Computation". In: *Application and Theory of Petri Nets and Concurrency*. Ed. by Wil van der Aalst and Eike Best. Cham: Springer International Publishing, 2017, pp. 139–158. ISBN: 978-3-319-57861-3.

[8] A. David et al. "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)*. Vol. 7214. LNCS. Springer-Verlag, 2012, 492–497.

[9] Martin Didriksen et al. "Automatic Synthesis of Transiently Correct Network Updates via Petri Games". In: *Application and Theory of Petri Nets and Concurrency*. Ed. by Didier Buchs and Josep Carmona. Cham: Springer International Publishing, 2021, pp. 118–137. ISBN: 978-3-030-76983-3.

[10] Serge Haddad and Jean-François Pradat-Peyre. "Efficient reductions for LTL formulae verification". In: *Paris, France* (2004).

[11] Serge Haddad and Jean-François Pradat-Peyre. "New Efficient Petri Nets Reductions for Parallel Programs Verification". In: *Parallel Process. Lett.* 16 (2006), pp. 101–116.

[12] Lasse Jacobsen et al. "Verification of Timed-Arc Petri Nets". In: *SOFSEM 2011: Theory and Practice of Computer Science*. Ed. by Ivana Černá et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 46–72. ISBN: 978-3-642-18381-2.

[13] Jonas F. Jensen et al. "TAPAAL and Reachability Analysis of P/T Nets". In: *Transactions on Petri Nets and Other Models of Concurrency XI*. Ed. by Maciej Koutny, Jörg Desel and Jetty Kleijn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 307–318. ISBN: 978-3-662-53401-4. DOI: 10.1007/978-3-662-53401-4_16. URL: https://doi.org/10.1007/978-3-662-53401-4_16.

[14] Jonas Finnemann Jensen et al. "TAPAAL and Reachability Analysis of P/T Nets". English. In: *Transactions on Petri Nets and Other Models of Concurrency XI*. Lecture Notes in Computer Science. Germany: Springer, 2016, pp. 307–318. ISBN: 978-3-662-53400-7. DOI: 10.1007/978-3-662-53401-4_16.

[15] Kurt Jensen. "Coloured Petri nets". In: *Petri Nets: Central Models and Their Properties: Advances in Petri Nets 1986, Part I Proceedings of an Advanced Course Bad Honnef, 8–19 September 1986*. Ed. by W. Brauer, W. Reisig and G. Rozenberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 248–299. ISBN: 978-3-540-47919-2. DOI: 10.1007/BFb0046842. URL: https://doi.org/10.1007/BFb0046842.

[16] F. Kordon et al. *Complete Results for the 2020 Edition of the Model Checking Contest*. http://mcc.lip6.fr/2021/results.php. 2021. (Visited on 2021).

[17] T. Murata. "Petri nets: Properties, analysis and applications". In: *Proceedings of the IEEE* 77.4 (1989), pp. 541–580. DOI: 10.1109/5.24143.

[18] Emmanuel Paviot-Adet et al. *LTL under reductions with weaker conditions than stutter-invariance*. 2021. arXiv: 2111.03342 [cs.CL].

[19] Carl Adam Petri. "Communication with automata". In: (1966).

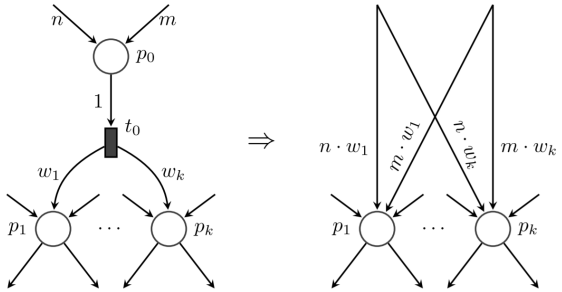
[20] Amir Pnueli. "The temporal logic of programs". In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32.

[21] Yann Thierry-Mieg. "Structural Reductions Revisited". In: *Application and Theory of Petri Nets and Concurrency*. Ed. by Ryszard Janicki, Natalia Sidorova and Thomas Chatain. Cham: Springer International Publishing, 2020, pp. 303–323. ISBN: 978-3-030-51831-8.

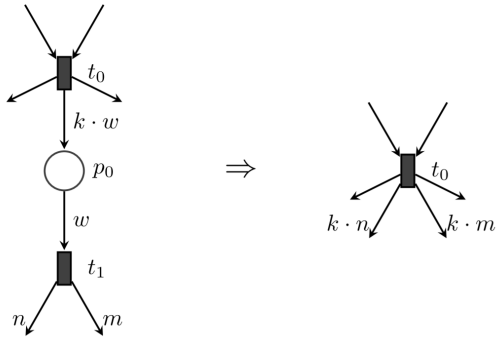
A Rules Cheat Sheet

The figures for rules A-H are taken from [5]. The figures for Rule I and Rule 10 were made for this appendix.

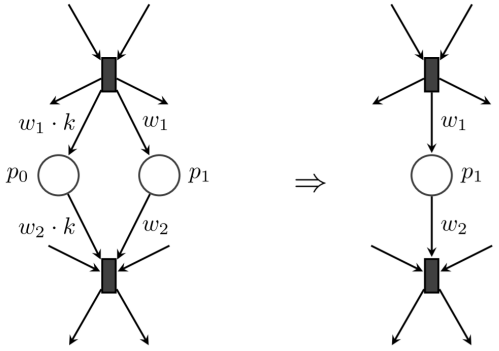
A Sequential transition removal: Merges place-transition pairs (p,t) where t can always be immediately fired when tokens enter p, into the transitions that add tokens to p.



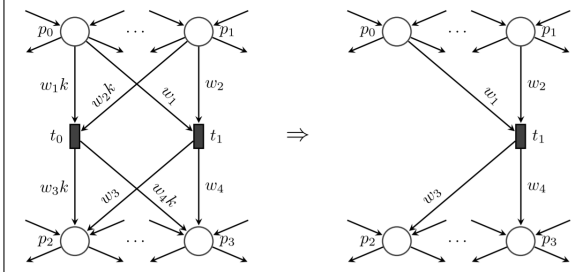
B Sequential place removal: Merges place-transition pairs (p,t) where t can always be immediately fired when tokens enter p, into the transitions that add tokens to p.



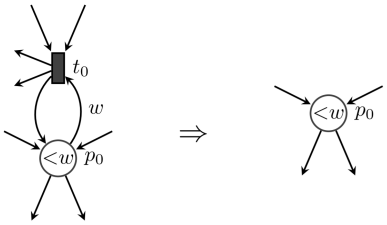
C Parallel place removal: Removes places that can never be the limiting factor on any behavior because of a parallel place.



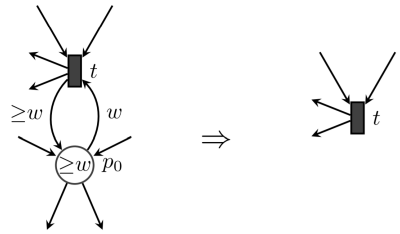
D Parallel transition removal: Removes transitions that have the same preconditions and effect as firing another transition n times.



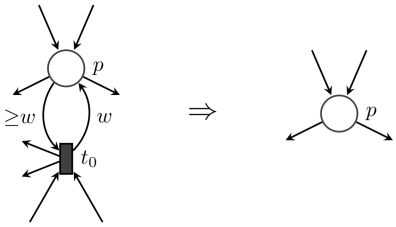
E Dead transition removal: Removes transitions that can never fire because its pre set cannot gain tokens.

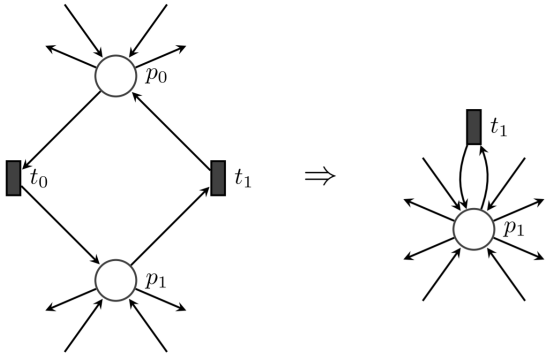
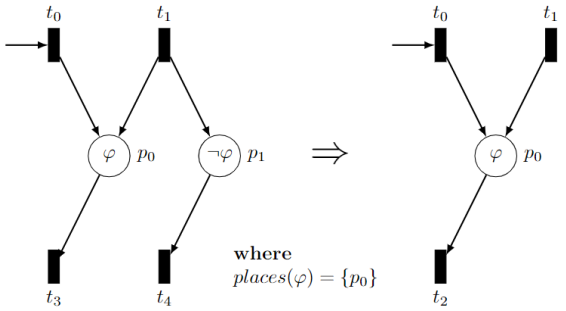
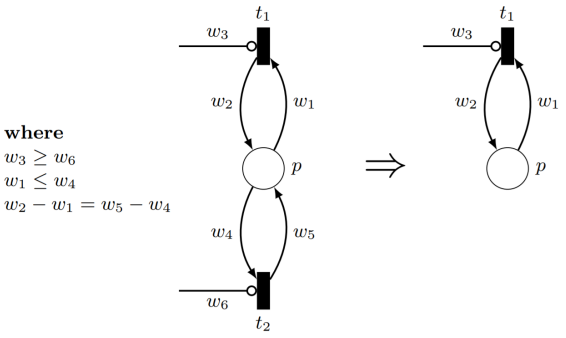


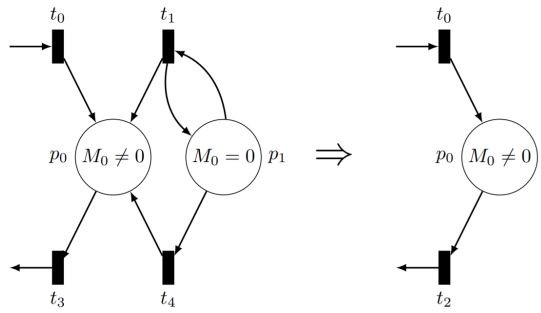
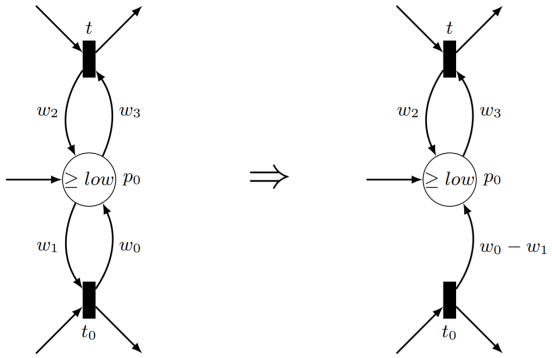
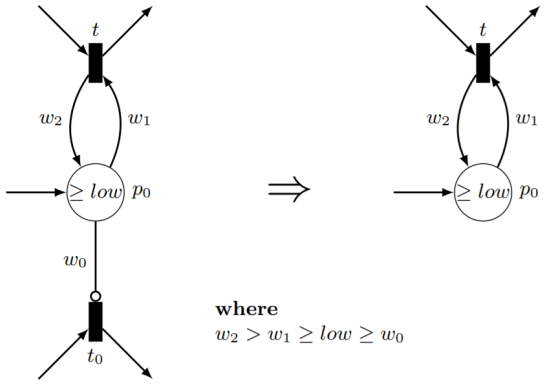
F Redundant place removal: Removes places that never disables its post set.

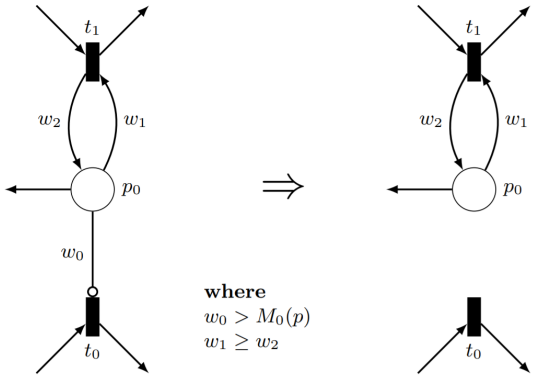
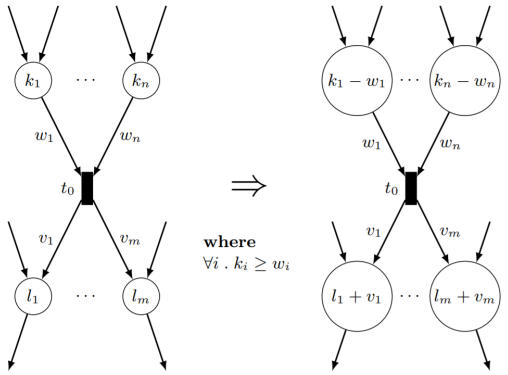
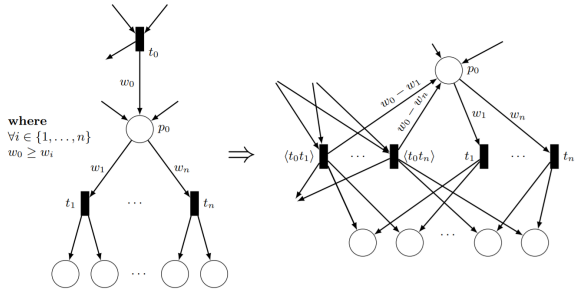


G Redundant transition removal: Removes transitions that only remove tokens from the net.



H	Simple cycle removal: Combines cycles of two places into a single place. 
I	Relevance: Removes any part of the net that does not enable behavior that can affect the query. 
L	Dominated transitions: Removes transitions that have the same effect, but more preconditions than another transition. 

M	Effectively dead places and transitions: Removes places that cannot gain or lose tokens and transitions that can never fire due to that. 
N	Redundant arcs: Removes arcs to places that always have the required number of tokens. 
O	Inhibited transitions: Removes transitions that are always inhibited. 

P	Redundant inhibitor arcs: Removes inhibitor arcs that never inhibit.  **where** $w_0 > M_0(p)$ $w_1 \geq w_2$
Q	Preemptive transition firing: Fires initially enabled transitions when it is safe to do so.  **where** $\forall i \,.\, k_i \geq w_i$
R	Atomic post-agglomerable producer: Post-agglomeration that removes interleavings, except only one producer at a time such that it supports weights. 

Rule 10	Maximal Unmarked Siphon: Removes places that do not contain tokens and can never gain tokens and all transitions dependent on them.