Petri Nets with Discrete Variables

Jonas Finnemann Jensen, Thomas Nielsen, Lars Kærlund Østergaard

Department of Computer Science, Aalborg University, Denmark

Abstract. Petri nets are a well-known graphical language for conceptual modelling. We propose a new model called Petri nets with discrete variables (PNDVs) that permits additional modelling convenience over classical Petri nets. We show that PNDVs are Turing complete and give a limited subset with the same expressive power as Petri nets. Moreover, we demonstrate that PNDVs can simulate bounded discrete timed-arc Petri nets. We apply heuristic algorithms for reachability analysis of PNDVs with a tool, we have developed called PeTe. Finally, we demonstrate the advantages of these algorithms with experimental results from using this tool.

1 Introduction

Model checking is a widely used technique for automatic verification of systems. This technique is used to verify the correctness of systems. Model-checking has also been applied to concurrent systems, where a widely-used model called Petri nets is used for the analysis and verification of these.

Petri nets (PNs) is a graphical language originally proposed by C.A. Petri in [16] for conceptual modelling of the flow of information in systems. Since then, there has been developed a number of classes of PNs to accommodate an increased need for descriptive power. Among these are timed-arc Petri nets (TAPNs) [9] that enable the specification and verification of real-time systems, as well as colored Petri nets (CPNs) [11], a versatile high-level type of PN which can be used for the specification, implementation and simulation of complex systems.

PNs can become complicated when expressing large systems, this is due to the limited set of basic constructs available in the language. For instance, some systems require additional constraints in order to ensure that the model is bounded, i.e. a bounded PN. A well-known modelling trick to ensure boundedness is using complementary-place transformation [3]. In conceptual modelling this construction may be undesirable due to the introduction of redundant information and artificial elements into the model, requiring users of the model to recognize and ignore redundancy. Moreover, the model can be harder to visualize, since simplicity is sacrificed in order to ensure boundedness.

We propose a new model called Petri nets with discrete variables (PNDVs) that primarily seeks to add modelling convenience and compactness to PNs, while at the same time ensuring that verification is possible. This model is a PN extended with a set of finite global integer variables, used in pre-conditions

and post-assignments on transitions. This provides a compact way of exploiting common encoding tricks used in PNs.

With the PNDV model, boundedness can be expressed in a more compact way with guards on transitions using discrete variables, hence redundant places are not required. In addition, inhibitor arcs can be simulated without the need for a new type of arc, nor the complementary place construction, as this can be expressed with guards on transitions in PNDVs. Besides these modelling capabilities it is straightforward to model systems which alter the state of variables through sequences of transitions.

Moreover, we present a reduction from bounded discrete timed arc Petri nets (DTAPNs) to PNDVs that demonstrates how PNDVs can be used for deciding marking reachability for DTAPNs.

Finally, we have developed a tool called PeTe (PeTri net Engine) for modelling and verification of PNDVs and PNs. This tool will be used to conduct reachability experiments with various heuristics. In addition, PeTe implements an algorithm for translating bounded DTAPNs into PNDVs. Experimental results show that the heuristics we have developed for reachability analysis are significantly faster than naive and randomized depth first search.

1.1 Related Work

The addition of variables in PNs has been done before with colored Petri nets (CPNs) that allow the execution of arbitrary program code when firing transitions, e.g. CPN Tools [17,11], a tool for simulating and analyzing CPNs, supports the functional programming language Standard ML.

The Petri net model we propose is different from CPNs, i.e. in PNDVs tokens are not colored (do not carry data), data structures are global and finite (and only permit integer variables), the guards imposed on transitions are restricted and do not allow for execution of code. It is well-known that the verification problems, such as reachability, coverability and boundedness, concerning CPNs are undecidable. Hence, model checking can be impossible for unbounded and even bounded CPNs [13]. PNDVs provides modelling convenience while guaranteeing decidability of the aforementioned model-checking problems for bounded nets. PNDVs have the same modelling power as PNs with inhibitor arcs. As we shall see later, we introduce p-free PNDVs, which correspond to ordinary PNs, but still with many of the modelling capabilities available.

We give a reduction from bounded DTAPNs to PNDVs for reachability analysis. In general reachability for DTAPNs is undecidable [18], however Escrig showed that finite timed reachability for unbounded DTAPNs is decidable by only simulating time up to some instant in [8]. Our approach works for bounded DTAPNs and uses a technique where tokens in each place are aged up to a maximal value. Then for delays larger than this value, tokens are no longer aged. This is similar to a technique used by Escrig in [18], where state graphs are used for reachability analysis on bounded TAPNs (real-valued time).

Outline: The next section will cover preliminaries for the rest of this paper. In Section 3 we introduce PNDVs. In Section 4 we introduce a subset of PNDVs

with the same expressive power as ordinary PNs. Section 5 describes a reduction from bounded DTAPNs to PNDVs for deciding reachability. In Section 6 algorithms for reachability analysis and experimental results are presented. Finally, in Section 7 we give conclusions and ideas for future work.

2 Preliminaries

Before we cover modelling and verification of PNDVs in more detail, we introduce the required terminology and definitions for PNs and the new PNDV model.

Let $\mathbb{N}$ denote the set of natural numbers including zero and $\mathbb{Z}$ the set of integers.

Definition 1. *(Petri net with inhibitor arcs) A Petri net with inhibitor arcs is a quadruple $N = (P, T, F, Inhb)$, where*

- *P is a finite set of places,*
- *T is a finite set of transitions, where $P \cap T = \emptyset$,*
- *$F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the flow function.*
- *$Inhb \subseteq P \times T$ is the set of inhibitor arcs, s.t. $(p, t) \in Inhb$ implies $F(p, t) = 0$.*

Let $N = (P, T, F, Inhb)$ be a PN. A marking is a mapping $M : P \to \mathbb{N}$ that assigns tokens to places. The set $\mathcal{M}(N)$ denotes the infinite set of all markings on N. If $Inhb = \emptyset$ then N is said to be an ordinary PN, for convenience we will write this as a triple $N = (P, T, F)$. A marked PN is a pair (N, M_0), where M_0 is an initial marking on N. The *preset* ${}^\bullet y$ for a place or transition y is defined as ${}^\bullet y = \{z \in P \cup T \mid F(z, y) > 0\}$, likewise, the *postset* is $y^\bullet = \{z \in P \cup T \mid F(y, z) > 0\}$. We denote the inhibitor places for a transition t as $Inhb(t) = \{p \mid (p, t) \in Inhb\}$. Given marking M, we say a transition t is *enabled* if it holds that

$$\forall p \in {}^\bullet t : F(p, t) \leq M(p) \wedge \forall p \in Inhb(t) : M(p) = 0$$

A transition t can *fire* if it is enabled, which leads to a new marking M', where $M'(p) = M(p) - F(p, t) + F(t, p)$ for every place $p \in P$. We write $M \xrightarrow{t} M'$ if from the marking M by firing t we reach the marking M'. We write $M \to M'$ if $M \xrightarrow{t} M'$ for some $t \in T$. The reflexive transitive closure of $\to$ is $\xrightarrow{*}$. Let $\mathcal{R}(M_0) = \{M' \mid M_0 \xrightarrow{*} M'\}$ denote the set of all reachable markings from initial marking M_0. A marking M' is said to be *reachable* from the initial marking M_0 if $M' \in \mathcal{R}(M_0)$.

We define a logic for reachability of PNs with meta-variables and syntactic categories for expressions $e_x \in \mathbf{Expr_x}$ and conditions $c_x \in \mathbf{Cond_x}$. The logic is defined as follows

$$e_x ::= z \mid p \mid e_x \oplus e_x, \quad \text{where } z \in \mathbb{Z},\ p \in P,\ \oplus \in \{+, -, *\} \tag{1}$$

$$c_x ::= e_x \bowtie e_x \mid c_x \vee c_x \mid c_x \wedge c_x \mid \neg c_x, \quad \text{where } \bowtie \in \{=, <, \leq, >, \geq, \neq\} \tag{2}$$

A marking M satisfies a condition c_x, denoted $M \models c_x$, if by replacing p in c_x with $M(p)$ for all $p \in P$ the formula evaluates to true. We say that a query

$c_x \in \mathbf{Cond_x}$ is *satisfiable* if there exists a reachable marking $M \in \mathcal{R}(M_0)$ s.t. $M \models c_x$.

PNs are illustrated as usual, where circles are places, tokens are black dots on places, transitions are black rectangles, input and output arcs are drawn as arrows, while inhibitor arcs are straight lines with a small circle at the end. An inscription on an arc denotes that its weight is greater than one. Figure 1 illustrates a PN simulating a computer scientist. A token in the place *inactive* means that the computer scientist is inactive, while a token in places *coffee machine* and *pizza* indicates that there is coffee and pizza available. The inhibitor arc between *pizza* and *grab coffee* prevents the computer scientist from grabbing a cup of coffee when pizza is available; leaving *eat pizza* the only transition enabled, when inactive. Once the pizza has been consumed, the transition *grab coffee* can be fired, allowing the computer scientist to work and output a paper.

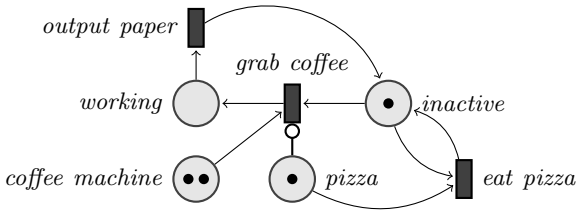


Fig. 1: A Petri net model of a computer scientist

3 Petri Nets with Discrete Variables

A Petri net with discrete variables (PNDV) is a PN extended with a finite set of integer variables, $X = \{x_1, \ldots, x_m\}$, and pre- and post-conditions on transitions. We define the meta-variables and syntactic categories for expressions $e \in \mathbf{Expr}$, conditions $c \in \mathbf{Cond}$ and assignments $a \in \mathbf{Assign}$.

The language describing expressions and conditions over X and places P is

$$e ::= x \mid z \mid p \mid e \oplus e, \quad \text{where } x \in X,\ z \in \mathbb{Z},\ p \in P,\ \oplus \in \{+, -, *\} \tag{3}$$

$$c ::= e \bowtie e \mid c \vee c \mid c \wedge c \mid \neg c, \quad \text{where } \bowtie \in \{=, <, \leq, >, \geq, \neq\} \tag{4}$$

$$a ::= (x_1 := e_1, x_2 := e_2, \ldots, x_m := e_m), \quad \text{where } e_1, \ldots, e_m \in \mathbf{Expr} \tag{5}$$

For convenience we may choose to write only the variables that are changed in an assignment, e.g. $a = (x_i := e_i, x_j := e_j)$ if only x_i, x_j are changed.

Definition 2. *(Petri net with discrete variables) A PNDV is a 7-tuple* $N = (P, T, X, \mathit{Range}, F, \mathit{Pre}, \mathit{Post})$ *, where* P*,* T *and* F *are as defined for PNs and*

- $X = \{x_1, \ldots, x_m\}$ *is a finite set of integer variables,*
- $\mathit{Range} : X \to \mathbb{N} \setminus \{0\}$ *assigns the maximum values for variables in* X*,*
- $\mathit{Pre} : T \to \mathbf{Cond}$ *is a mapping from transitions to pre-conditions, and*
- $\mathit{Post} : T \to \mathbf{Assign}$ *maps transitions to post-assignments.*

Let $N = (P, T, X, \mathit{Range}, F, \mathit{Pre}, \mathit{Post})$ be a PNDV. A valuation is a function $V : X \to \mathbb{N}$, where $V(x) \leq \mathit{Range}(x)$ for all $x \in X$. All valuations on N are denoted $\mathcal{V}(N)$. A state on N is a pair $S = (M, V) \in \mathcal{M}(N) \times \mathcal{V}(N)$, where M is a marking and V is a valuation. The set of all states on N is defined as $\mathcal{S}(N) = \mathcal{M}(N) \times \mathcal{V}(N)$. A marked PNDV is a pair (N, S_0), where $S_0 = (M_0, V_0)$ is the initial state on N and $V_0(x) = 0$ for all $x \in X$. We say that a state $S = (M, V)$ satisfies a pre-condition $c \in \mathbf{Cond}$, denoted $S \models c$, if by replacing the places and variables in c with the corresponding values from M and V, the formula evaluates to *true*. Given a state S, an expression $e \in \mathbf{Expr}$ evaluates to a new value $V \in \mathbb{Z}$, denoted $V = \mathit{eval}(e, S)$. An assignment $a = (x_1 := e_1, \ldots, x_m := e_m) \in \mathbf{Assign}$ evaluates to a new valuation $V' = \mathit{eval}(a, S)$, where for all $i : 1 \leq i \leq m$

$$\mathit{eval}(a, S)(x_i) = \mathit{eval}(e_i, S) \bmod (\mathit{Range}(x_i) + 1). \tag{6}$$

Note that $\mathit{Range}(x_i)$ is the largest value x_i can have. Let $S = (M, V)$ be a state on N and $t \in T$ a transition, we say that t is *enabled* in S if it holds that

$$\forall p \in {}^\bullet t : F(p, t) \leq M(p) \wedge S \models \mathit{Pre}(t) \tag{7}$$

If t is enabled in S, it can fire, which leads to a new state S', such that the resulting state obtained by firing t is $S' = (M', \mathit{eval}(\mathit{Post}(t), S))$, where $M'(p) = M(p) - F(p, t) + F(t, p)$ for every place $p \in P$. The transition relation of PNDVs is similar to that of PNs. We write $S \xrightarrow{t} S'$ if by firing t from state S we reach state S'. We write $S \to S'$ if $S \xrightarrow{t} S'$ for some $t \in T$. The reflexive transitive closure of $\to$ is $\xrightarrow{*}$. Let $\mathcal{R}(S_0) = \{S' \mid S_0 \xrightarrow{*} S'\}$ denote the set of all reachable states from the initial state S_0. A state S' is said to be reachable from the initial state S_0 if $S' \in \mathcal{R}(S_0)$.

Similar to PNs, when querying reachability on PNDVs we use the **Cond** language. We say that a query $c \in \mathbf{Cond}$ is satisfiable if there exists a reachable state $S \in \mathcal{R}(S_0)$ s.t. $S \models c$.

3.1 Producer-Consumer Example

Now we shall see a variant of a producer-consumer (PC) system modelled with a PNDV. A PC system usually contains at least one producer process and a consumer process. The producer delivers objects to the consumer. To avoid overwhelming the consumer, objects are placed into a bounded buffer. A variation of a PC system is shown in Figure 2 with an extra producer, where pre- and post-conditions are shown as labels immediately next to transitions prefixed with *pre:* and *post:*, respectively. The consumer receives objects from two producers taking turns to add objects, i.e. tokens, to the buffer. The variable B indicates which producer is active. Since there are two producers, $B = 0$ or $B = 1$, i.e. $Range(B) = 1$. To ensure boundedness, the *buffer* place can hold a maximum of two tokens. For a producer to output a token to the buffer, the *buffer* place must contain less than two tokens and B must have the value corresponding to its pre-condition. When *take* is fired the active producer is swapped (recall that the value of B wraps around when overflow happens).

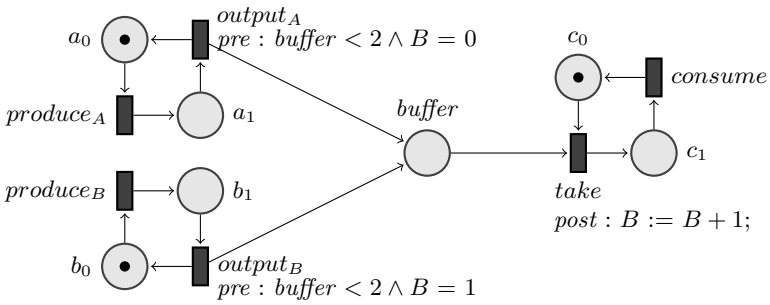


Fig. 2: A PNDV model of a producer-consumer system

3.2 Expressiveness of the PNDV Model

Here we will discuss the expressiveness of PNDVs. Recall that the language defined in Section 3 allows us to construct conditions that include places. The motivation for enabling these kinds of conditions is a convenient way of testing for zero, similar to the way inhibitor arcs work. However, PNs with inhibitor arcs are Turing complete [15], consequently reachability, coverability and boundedness are undecidable.

Theorem 1. *PNDVs have full Turing power.*

Proof. We can reduce an PN with inhibitor arcs $N = (P, T, F, Inhb)$ into a PNDV $N' = (P, T, \emptyset, Range, F, Pre, Post)$, where $Pre(t) = \{\bigwedge_{p \in Inhb(t)} p = 0\}$ for all $t \in T$. Clearly, the labelled transition systems of N and N' are isomorphic, since the pre-condition for a transistion t is only satisfied if every place $p \in Inhb(t)$ is empty. Thus, PNDVs have full Turing power, since they can simulate PNs with inhibitor arcs. □

4 P-Free PNDV

We introduce a subclass of PNDVs with computational power equivalent to that of a PN. We call this subclass p-free, since places cannot be referenced in conditions and assignments. We define p-free expressions $\mathbf{Expr_p}$ as the subset of expressions $\mathbf{Expr}$ where places $p \in P$ do not occur. Similarly, we define p-free conditions $\mathbf{Cond_p} \subset \mathbf{Cond}$ and p-free assignments $\mathbf{Assign_p} \subset \mathbf{Assign}$ as the respective subsets where p does not occur.

Definition 3. *A PNDV $N = (P, T, X, Range, F, Pre, Post)$ is p-free if $Pre(t) \in \mathbf{Cond_p}$ and $Post(t) \in \mathbf{Assign_p}$ for all $t \in T$.*

Since a p-free condition $c_p \in \mathbf{Cond_p}$ cannot reference any places, it can be evaluated given only a valuation V. Likewise, a p-free assignment $a_p \in \mathbf{Assign_p}$ can be evaluated given a valuation V, denoted $V' = eval(a_p, V)$, defined as in Equation 6 with $eval(e_i, V)$, instead of $eval(e_i, S)$.

A p-free PNDV $N = (P, T, X, Range, F, Pre, Post)$ can be translated into a PN $N' = (P', T', F')$ by creating new bounded places to simulate the variables. This is done by creating places p_x and complement places $\overline{p_x}$ for each variable $x \in X$. The number of tokens in place p_x is the value of x and place $\overline{p_x}$ bounds the place s.t. $p_x + \overline{p_x} = Range(x)$. A transition t_V is introduced in N' for each valuation V satisfying $Pre(t)$ for every transition $t \in T$.

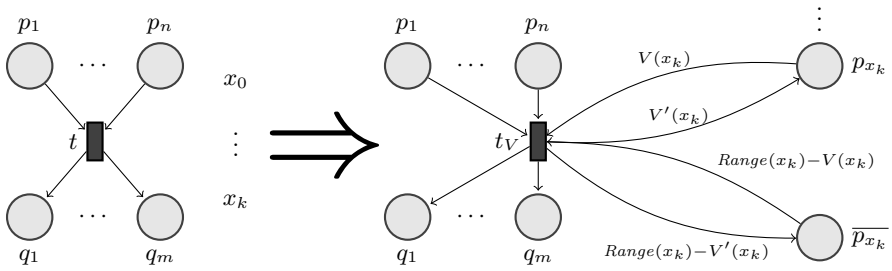


Fig. 3: A reduction from p-free PNDV to PN, where $V' = eval(Post(t), V)$

Figure 3 shows how two places are created for each variable in the p-free PNDV and how these are connected to a transition t_V, which the translated net may have exponentially many of. The pseudo code for this translation is listed in Algorithm 1.

Remark 1. Given a p-free PNDV $N = (P, T, X, Range, F, Pre, Post)$ the resulting PN $N' = \text{P-FREE-PNDV-TO-PN}(N)$, from translation using Algorithm 1, can be exponential in the size of N. Algorithm 1 creates $|P| + 2 \cdot |X|$ places, but the number of transitions is bounded by $O(|T| \cdot \prod_{x \in X} Range(x))$.

Algorithm 1 Conversion from p-free PNDV to PN

function p-free-PNDV-to-PN(N)
  $(P, T, X, Range, F, Pre, Post) = N$
  $P' = P \cup \{p_x, p'_x \mid x \in X\}$
  $T' = \emptyset$
  for all $V \in \mathcal{V}(N)$ **do** ▷ Consider all valuations
    for all $t \in T$ **do**
      if $V \models Pre(t)$ **then**
        $T' = T' \cup \{t_V\}$
        for all $p \in P$ **do** ▷ Connect to same places as t
          $F'(p, t_V) = F(p, t)$
          $F'(t_V, p) = F(t, p)$
        end for
        $V' = eval(Post(t), V)$
        for all $x \in X$ **do**
          $F'(p_x, t_V) = V(x)$ ▷ Depend on the assumed valuation
          $F'(p'_x, t_V) = Range(x) - V(x)$
          $F'(t_V, p_x) = V'(x)$ ▷ Output the resulting valuation
          $F'(t_V, p'_x) = Range(x) - V'(x)$
        end for
      end if
    end for
  end for
  return (P', T', F')
end function

Definition 4. *Let $N_1 = (P_1, T_1, X_1, Range_1, F_1, Pre_1, Post_1)$ be a p-free PNDV and $N_2 = (P_2, T_2, F_2) =$ p-free-PNDV-to-PN(N_1) be the PN translation of N_1 using Algorithm 1. Let $S_1 = (M_1, V_1)$ on N_1 and marking M_2 on N_2. We say that S_1 corresponds to M_2, denoted $S_1 \equiv M_2$, if $M_1(p) = M_2(p)$ for all $p \in P$ and $V_1(x) = M_2(p_x) = Range(x) - M_2(\overline{p_x})$ for all $x \in X$.*

Theorem 2. *The labelled transition system (LTS) of a marked p-free PNDV (N_1, S_1) is isomorphic to the LTS of its translated marked PN (N_2, M_2), where $N_2 =$ p-free-PNDV-to-PN(N_1) and any M_2 s.t. $S_1 \equiv M_2$.*

A proof of Theorem 2 is provided in Appendix A.

Corollary 1. *Reachability is decidable for p-free PNDVs.*

Proof. Let (N_1, S_1) be a marked p-free PNDV, then following Theorem 2, it is possible to construct a marked PN (N_2, M_2) s.t. its LTS is isomorphic to the LTS of (N_1, S_1). Since reachability is decidable for PNs [14,12], it follows that reachability is deciable for p-free PNDVs. □

5 Discrete Timed-Arc Petri Nets

In this section we shall describe a reduction from bounded (DTAPNs), with discrete time semantics, to PNDVs that preserves marking reachability. This reduction demonstrates the full modelling capabilities of PNDVs by showing that this model can encode bounded DTAPNs for reachability analysis.

Definition 5. *(Discrete timed-arc Petri net) A DTAPN is a quadruple* $D = (P, T, F, \mathit{times})$ *, where* P *and* T *are defined as for PNs and*

- $F \subseteq (P \times T) \cup (T \times P)$ *is the flow relation, and*
- $\mathit{times} : F|_{P \times T} \to \{[a, b] \mid a \in \mathbb{N}, b \in \mathbb{N} \cup \{\infty\}\}$ *maps intervals to input arcs.*

Let $D = (P, T, F, \mathit{times})$ be a DTAPN. A marking on D is a mapping $M : P \to \mathcal{B}(\mathbb{N})$, where $\mathcal{B}(\mathbb{N})$ denotes the set of finite multisets of natural numbers. The natural numbers correspond to the age of the tokens at a given place. The preset ${}^\bullet y$ of a place or transition y is ${}^\bullet y = \{z \in P \cup T \mid (z, y) \in F\}$, the postset is defined similarly. A marked DTAPN is a pair (D, M_0) where D is a DTAPN and M_0 is an initial marking on D s.t. all tokens have age 0.

Given a marking M, a transition t is enabled if there exists a token $x \in M(p)$ where $x \in \mathit{times}(p, t)$, for all $p \in {}^\bullet t$. If t is enabled in marking M then it can fire, yielding a new marking M', denoted $M \xrightarrow{t} M'$, where $M'(p) = (M(p) \setminus \mathit{In}(p, t)) \cup \mathit{Out}(t, p)$ for every place $p \in P$, where $\setminus$ and $\cup$ are operations on multisets,

$$\mathit{In}(p,t) = \begin{cases} \{x\} & \text{if } p \in {}^\bullet t \wedge x \in M(p) \wedge x \in \mathit{times}(p,t) \\ \emptyset, & \text{otherwise} \end{cases}$$

$$\mathit{Out}(p,t) = \begin{cases} \{0\}, & \text{if } p \in t^\bullet \\ \emptyset, & \text{otherwise} \end{cases}$$

and from each place $p \in {}^\bullet t$, a single token satisfying the age constraint is removed and a new token of age 0 is added to every place $p \in t^\bullet$. Given a marking M a delay can occur yielding a new marking M', denoted $M \xrightarrow{d} M'$, where $M'(p) = \{x + 1 \mid x \in M'(p)\}$ for all $p \in P$. This increments the age of all tokens by one. For simplicity we assume that a time delay is always 1, as any other delay can be simulated with this.

A marked DTAPN (D, M_0) is said to be k-bounded if it holds that $|M(p)| \leq k$ for every place $p \in P$ in every reachable marking $M \in \mathcal{R}(M_0)$. For convenience we may write $\beta_i(p, t)$ to denote b_i, for $i = 1, 2$, when $\mathit{times}(p, t) = [b_1, b_2]$.

5.1 Reduction From Bounded DTAPN to PNDV

Reachability is known to be undecidable for discrete timed-arc Petri nets DTAPNs [18]. Nevertheless, reachability is decidable for bounded DTAPNs [5]. In this section we propose a reduction from k-bounded DTAPN D to PNDV N

that preserves reachability. The overall idea of the reduction is to simulate aging of up to k tokens in every place and to ensure that time intervals are simulated correctly. Reduction from a timed model into an untimed model introduces a number of challenges, presented in the following.

The first problem is how markings on a DTAPN are represented in a PNDV, since a marking in D associates ages with individual tokens, whereas N only stores the number of tokens in each place. This problem is solved by storing the ages of up to k tokens with k variables $\{x_p^1, \ldots, x_p^k\}$ for every place $p \in P$.

For example, a marking M on D, where $M(p) = \{0, 2, 2\}$ is represented by a state $S' = (M', V')$ on N, s.t. $M'(p) = 3$, $V'(x_p^1 = 0)$, $V'(x_p^2 = 2)$ and $V'(x_p^3 = 2)$. Notice that $M'(p)$ is the number of tokens in p and the variables x_p^1, x_p^2 and x_p^3 keep track of the age of each token.

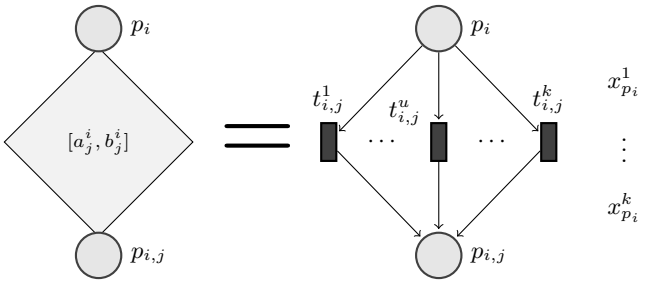
For every place p its token variables $\{x_p^1, \ldots, x_p^k\}$ are bounded by a maximal value, $\textsc{MaxAge}(p)$, s.t. enabledness of every transition $t \in p^\bullet$ is unaffected by further aging [18]. $\textsc{MaxAge}(p)$ is computed by adding 1 to the maximal interval endpoint that is not ∞ on the output arcs from p.

$$\textsc{MaxAge}(p) = 1 + max\{\beta_i(p,t) \mid t \in p^\bullet, \beta_i(p,t) < \infty, i = 1, 2\} \tag{8}$$

To simulate transitions we could introduce a transition for every possible state, but to avoid exponentially many transitions we simulate one transition firing $M \to M'$ in D with multiple transition firings $S \to S_1 \to \cdots \to S_q \to S'$. We call the intermediate states $S_1, \ldots, S_q$ *unstable* and introduce a lock variable ℓ. The lock ensures that we avoid undesirable and inconsistent behavior, for instance, if a token was consumed while aging.

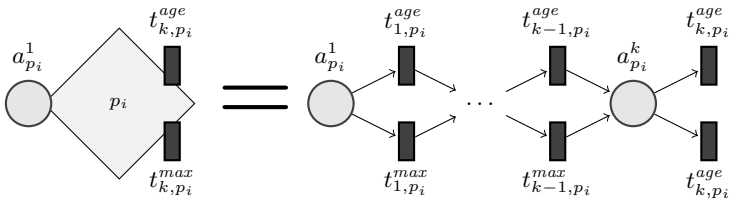
To model transitions in D we insert an interval gadget between every transition and its input places. Figure 4 illustrates an interval gadget that simulates an input arc (p_i, t_j) for transition t_j. Token variables for input place p_i are shown to the right. The take-token transitions $t_{i,j}^u$, for $1 \leq u \leq k$, simulate consumption of token $x_{p_i}^u$, and are only enabled if there are at least u tokens in p_i and token $x_{p_i}^u$ satisfies the time interval, since the number of tokens in p_i denotes the number of variables to consider for consumption. When token $x_{p_i}^u$ is removed, the post-condition ensures that the ages of the remaining tokens are preserved. The lock ℓ is also acquired, guaranteeing that no other transition can begin firing before t_j has finished.

To simulate delays we construct a 1-safe ring of aging gadgets, s.t. all variables have been incremented by 1 (if the variables are less than their maximum values) when the token has done a single pass through the ring. Note that the aging ring also acquires the lock ℓ, such that tokens cannot be consumed while aging. Figure 5 shows the aging gadget for place p_i. An age transition t_{u,p_i}^{age} increments the token variable $x_{p_i}^u$ by 1, and is enabled if $x_{p_i}^u < \textsc{MaxAge}(p_i)$. The max transition t_{u,p_i}^{max} does not increment the token variable, and is enabled when no further aging is possible, i.e. $x_{p_i}^u = \textsc{MaxAge}(p_i)$. Note, t_{u,p_i}^{age} and t_{u,p_i}^{max} are never enabled at the same time. The aging gadgets for all places are arranged in a ring, where the first place is marked. The first and last transitions acquire and release the lock ℓ, see Algorithm 2 step 6 to 8 for details.



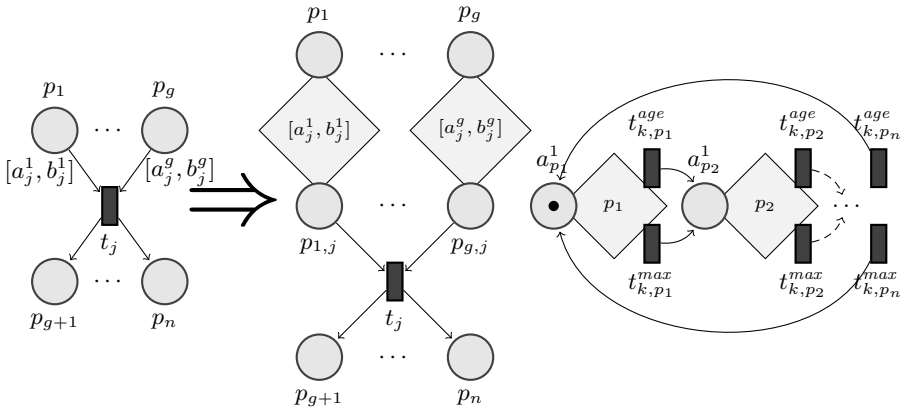
$$Pre(t_{i,j}^u) = (p_{i,j} = 0 \wedge p_i \geq u) \wedge (a_j^i \leq x_{p_i}^u \wedge x_{p_i}^u \leq b_j^i) \wedge (\ell = 0 \vee \ell = j)$$
$$Post(t_{i,j}^u) = \ell := j; \cup \{x_{p_i}^v := x_{p_i}^{v+1} \mid u \leq v < k\}$$

Fig. 4: An interval gadget $[a_j^i, b_j^i]$. Here p_i is some place $p_i \in {}^\bullet t_j$, shown in Figure 6



$$Pre(t_{u,p_i}^{max}) = x_{p_i}^u = \textsc{MaxAge}(p_i) \qquad Pre(t_{u,p_i}^{age}) = x_{p_i}^u < \textsc{MaxAge}(p_i)$$
$$Post(t_{u,p_i}^{age}) = x_{p_i}^u := x_{p_i}^u + 1;$$

Fig. 5: An aging gadget for one place with at most k tokens



$$Post(t_j) = \ell := 0; \cup \{x_p^u := x_{p_i}^{u-1}; \mid p_i \in t_j^\bullet, 2 \leq u \leq k\} \cup \{x_{p_i}^1 := 0; \mid p_i \in t_j^\bullet\}$$

Fig. 6: A reduction from a k-bounded DTAPN to PNDV. The diamond shapes represents either interval or aging gadgets. The aging ring is shown to the right

Figure 6 illustrates the reduction from k-bounded DTAPN to PNDV. When transition t_j fires, it releases the lock ℓ and shifts the token variables of the places in its postset, while setting the first token variable to 0. Note that it is possible to

deadlock in a situation where one of the interval gadgets for t_j has acquired the lock ℓ, but the other interval gadgets for the transition were unable to consume a token, so t_j cannot fire. However, as previously mentioned this reduction only preserves marking reachability, and not liveness. A formal description of this reduction is available in Algorithm 2.

Algorithm 2 Reduction from k-bounded DTAPN to PNDV

Input: A k-bounded DTAPN $D = (P, T, F, \mathit{times})$ with initial marking M_0, where $P = \{p_1, \dots p_i, \dots, p_n\}$ and $T = \{t_1, \dots t_j, \dots, t_m\}$.

Output: A PNDV $N = (P', T', X, F', \mathit{Range}, \mathit{Pre}, \mathit{Post})$ with initial state $S_0 = (M'_0, V_0)$.

1. Create original places and transitions, set $P' = P$ and $T' = T$
2. Create variables, $X = \{\ell\} \cup \{x^u_{p_i} \mid p_i \in P, 1 \leq u \leq k\}$, where $\mathit{Range}(\ell) = m + 1$ and $\mathit{Range}(x^u_{p_i}) = \textsc{MaxAge}(p_i)$ for all $x^u_{p_i} \in X \setminus \{\ell\}$.
3. For each $t_j \in T$, release lock and set the first token variable to 0 and shift the others, for all $p_i \in t_j^\bullet$
   $\mathit{Post}(t_j) = \ell := 0; \cup \{x^u_p := x^{u-1}_{p_i}; \mid p_i \in t_j^\bullet, 2 \leq u \leq k\} \cup \{x^1_{p_i} := 0; \mid p_i \in t_j^\bullet\}$.
4. For each output arc $(t_j, p_i) \in F$, set $F'(t_j, p_i) = 1$.
5. For each input arc $(p_i, t_j) \in F$, create an interval gadget (see Figure 4):
   (a) Add intermediate place, set $P' = P' \cup \{p_{i,j}\}$.
   (b) Connect it to t_j, set $F'(p_{i,j}, t_j) = 1$.
   (c) Add take-token transitions for interval $[a^i_j, b^i_j] = \mathit{times}(p_i, t_j)$, set $T' = T' \cup \{t^u_{i,j} \mid 1 \leq u \leq k\}$ with pre- and post conditions:
      – $\mathit{Pre}(t^u_{i,j}) = (p_{i,j} = 0 \wedge p_i \geq u) \wedge (a^i_j \leq x^u_{p_i} \wedge x^u_{p_i} \leq b^i_j) \wedge (\ell = 0 \vee \ell = j)$,
      – $\mathit{Post}(t^u_{i,j}) = \ell := j; \cup \{x^v_{p_i} := x^{v+1}_{p_i} \mid u \leq v < k\}$.
   (d) Connect these, set $F'(p_i, t^u_{i,j}) = F'(t^u_{i,j}, p_{i,j}) = 1$, for $1 \leq u \leq k$.
6. For each place $p_i \in P$ create an aging gadget (see Figure 5):
   (a) Add aging places, set $P' = P' \cup \{a^u_{p_i} \mid 1 \leq u \leq k\}$.
   (b) Add aging transitions, set $T' = T' \cup \{t^{age}_{u,p_i}, t^{max}_{u,p_i} \mid 1 \leq u \leq k\}$.
   (c) Connect aging places and transitions, set
      – $F'(a^u_{p_i}, t^w_{u,p_i}) = 1$, for $1 \leq u \leq k$,
      – $F'(t^w_{u,p_i}, a^{u+1}_{p_i}) = 1$, for $1 \leq u \leq k-1$, where $w \in \{\mathit{age}, \mathit{max}\}$.
   (d) Create pre- and post-conditions:
      – $\mathit{Pre}(t^{age}_{u,p_i}) = x^u_{p_i} < \textsc{MaxAge}(p_i)$,
      – $\mathit{Pre}(t^{max}_{u,p_i}) = x^u_{p_i} = \textsc{MaxAge}(p_i)$,
      – $\mathit{Post}(t^{age}_{u,p_i}) = x^u_{p_i} := x^u_{p_i} + 1$;
7. Connect the aging gadgets in a ring:
   – $F'(t^k_{p_i,w}, a^1_{p_{i+1}}) = 1$, for $1 \leq i \leq n-1$,
   – $F'(t^k_{p_n,w}, a^1_{p_1}) = 1$, where $w \in \{\mathit{age}, \mathit{max}\}$
8. Create start and stop conditions for aging ring:
   – $\mathit{Post}(t^{age}_{k,p_n}) = \ell := 0;\ x^u_{p_i} := x^u_{p_i} + 1$;
   – $\mathit{Post}(t^{max}_{k,p_n}) = \ell := 0$;
   – $\mathit{Post}(t^{age}_{1,p_1}) = \ell := m + 1;\ x^u_{p_i} := x^u_{p_i} + 1$;
   – $\mathit{Post}(t^{max}_{1,p_1}) = \ell := m + 1$;
9. Create initial marking, s.t. $M'_0(a^1_{p_1}) = 1$ and $M'_0(p_i) = |M(p_i)|$ for all $p_i \in P$.

5.2 Correctness of the Reduction

In this section we prove the correctness of the reduction. We argue that there is a correspondence between markings in the original net and the translation, using a technique similar to the method for relating timed transition systems, proposed in [6] and [10]. For the rest of this section let (D, M_0) be a marked k-bounded DTAPN and (N, S_0) be a marked PNDV translation of (D, M_0) using the reduction.

Definition 6 (Stable state). *A state $S = (M, V)$ on N is said to be stable, denoted $S \models stable$, if $V(\ell) = 0$.*

Definition 7 (Correspondence). *Let $M \in \mathcal{M}(D)$ and $S \in \mathcal{S}(N)$, M and $S = (M', V')$ are said to correspond, denoted $M \equiv S$, if $S \models stable$ and for all $p \in P$ it holds that*

$$\{min(x, MaxAge(p)) \mid x \in M(p)\} = \{V'(x_p^i) \mid 1 \leq i \leq M'(p)\}$$

, where the left- and right-handside are multisets.

Let $\rightsquigarrow$ denote a sequence of transitions $S = S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_{m-1} \rightarrow S_m = S'$ in N', where $S, S' \models stable$ and all intermediate states $S_1, S_2, \ldots, S_{m-1} \not\models stable$. The reflexive and transitive closure of $\rightsquigarrow$ is $\overset{*}{\rightsquigarrow}$. The function $tr : \mathbf{Cond_x} \rightarrow \mathbf{Cond}$ translates conditions, such that $tr(c_x) = c_x \wedge (\ell = 0)$ for all $c_x \in \mathbf{Cond_x}$.

Lemma 1. *Let M be a marking on D and S be a state on N, where $M \equiv S$ and $\alpha = \{t, d\}$.*

1. *If $M \xrightarrow{\alpha} M'$, then $S \rightsquigarrow S'$ such that $M' \equiv S'$.*
2. *If $S \rightsquigarrow S'$, then $M \xrightarrow{\alpha} M'$ such that $M' \equiv S'$.*

Proof (Sketch). If $M \xrightarrow{d} M'$, then an iteration through the aging ring $S \rightsquigarrow S'$ will increment all token variables, if needed. And because of the *max* application on the left handside in Definition 7 it does not matter that token variables are not aged beyond MaxAge(p) for their place p. If $M \xrightarrow{t_j} M'$, then correspondence between M and S ensures that $p_{i,j}$ can be marked for all i s.t. $p_i \in {}^\bullet t_j$, thus ensuring that t_j can fire in N , leading to a state S' corresponding to M'. Thus, we have shown the first case 1.

If $S \rightsquigarrow S'$, then one of two things could have happened, *(i)* N took an iteration through the aging ring, or *(ii)* a transition t_j was fired setting $\ell = 0$. In case (i) a delay $M \xrightarrow{d} M'$ will increment all token ages by one, again aging beyond MaxAge(p) for their respective places p will not affect correspondence. If some transition t_j was fired, then clearly tokens satisfying intervals on the arcs from the preset of t_j in D, s.t. $M \xrightarrow{t_j} M'$, must be available in M. Since the interval gadgets ensure that only tokens satisfying the interval on an input arc can be consumed by t_j. Thus, 2 holds for both case (i) and (ii). □

Lemma 2 (Agreement). *Let M be a marking on D, S be a state on N, where $M \equiv S$, and $c_x \in \mathbf{Cond_x}$, then $M \models c_x$ if and only if $S \models tr(c_x)$.*

Proof. By Definition 7, $M \equiv S$ implies that there is the same number of tokens in each place. Since the condition c_x only quantifies over the number of tokens, it must hold. Note that the definition of correspondence also ensures that $S = (M', V')$ is stable, thus $V'(\ell) = 0$. □

Theorem 3. *Let M be a marking on D and S be a state on N, where $M \equiv S$. Let $c_x \in \mathbf{Cond_x}$ be a query, then $M \xrightarrow{*} M'$ s.t. $M' \models c_x$ if and only if $S \xrightarrow{*} S'$ s.t. $S' \models tr(c_x)$.*

Proof. In step 9 of Algorithm 2 state $S_0 = (M'_0, V_0)$ and marking M'_0 are given the same number of tokens in each place, since M_0 and $V_0(x) = 0$ for all $x \in X$. By definition, the ages of tokens in a marked DTAPN are initially 0, meaning that $S_0 \equiv M_0$.

"⇒": Since $S_0 \equiv M_0$ we can conclude that $M \xrightarrow{*} M'$ if and only if $S \rightsquigarrow^{*} S'$ where $M' \equiv S'$ by repeated application of Lemma 1. Following Lemma 2 we prove that if $M \xrightarrow{*} M'$ where $M' \models c_x$, then $S \xrightarrow{*} S'$ where $S_0 \models tr(c_x)$. This is possible because $S \rightsquigarrow^{*} S'$ implies $S \xrightarrow{*} S'$.

"⇐": To prove that if $S \xrightarrow{*} S'$, where $S' \models tr(c_x)$ then $M \xrightarrow{*} M'$ where $M' \models c_x$ holds, we must consider the fact that S' must be stable in order to satisfy $tr(c_x)$, thus we might as well write $S \rightsquigarrow^{*} S'$ for which we know it holds. □

6 Verification of PNDVs

Now that we have demonstrated the modelling capabilities of PNDVs, we move onto the verification of this model. In this section we present algorithms for reachability analysis of bounded PNDVs. We describe a heuristic used to improve the efficiency of state space search, as well as an over-approximation technique that is useful for disproving reachability. Lastly, we present experimental results for these algorithms.

Algorithm 3 shows a general (naive) graph searching algorithm for performing reachability analysis. It can be implemented as a depth first search (DFS) or breadth first search (BFS), depending on the order in which states are taken from the queue in line 5. The algorithm guaranteed to terminate on bounded PNDVs. We can easily improve this algorithm by transforming it into a best first search algorithm (BestFS), by prioritizing the most promising states, using a heuristic cost estimate, described in Section 6.1.

Algorithm 3 General Reachability Search algorithm for satisfying a query

```
1: function REACHABILITY-SEARCH(N, S_0, q)
2:     (P, T, X, Range, F, Pre, Post) = N
3:     Q = Z = {S_0}
4:     while Q ≠ ∅ do
5:         Choose S from Q                                  ▷ Choose a state from Q
6:         if eval(q, M, V) = true, where S = (M, V) then
7:             return "Query q is satisfiable"
8:         end if
9:         for S', such that S -t-> S', where t ∈ T do      ▷ For each successor state
10:            if S' ∉ Z then
11:                Z = Z ∪ {S'}
12:                Q = Q ∪ {S'}                             ▷ Insert S' into the queue
13:            end if
14:        end for
15:        Q = Q \ {S}
16:    end while
17:    return "Query q is not satifisable"
18: end function
```

6.1 Best First Reachability Search

In this section we present a modification of Algorithm 3 that attempts to reach a marking satisfying the query using as few iterations as possible. Thus, if a query is satisfiable, this approach may explore a smaller subset of the state space than the naive approach in Algorithm 3.

The idea is to do best first search with a heuristic to choose a state S from the queue Q that is likely to be close to a state satisfying the query. In order to do this, we need a heuristic distance function to estimate the distance a state is from satisfying a query. The heuristic takes both logical conditions and comparisons into consideration when calculating the estimate.

To estimate the distance when comparing two integers, given an operator they need to satisfy, we introduce the auxillary function $\Delta : \mathbb{N} \times \{=, <, \leq, >, \geq, \neq\} \times \mathbb{N} \to \mathbb{N}$, as defined in Table 1.

$$\Delta(v_1, =, v_2) = |v_1 - v_2|$$
$$\Delta(v_1, \neq, v_2) = \begin{cases} 1, & \text{if } v_1 = v_2 \\ 0, & \text{otherwise} \end{cases}$$
$$\Delta(v_1, <, v_2) = max(v_1 - v_2 + 1, 0) \quad \Delta(v_1, >, v_2) = \Delta(v_2, <, v_1)$$
$$\Delta(v_1, \leq, v_2) = max(v_1 - v_2, 0) \quad \Delta(v_1, \geq, v_2) = \Delta(v_2, \leq, v_1)$$

Table 1: Formal Δ Specification

The function Δ is designed to yield a lower value if the numbers are close to satisfy the operator. For example, $\Delta(3, <, 2) = 2 > \Delta(3, <, 3) = 1$, because $\Delta(3, <, 3)$ is closer to being satisfied than $\Delta(3, <, 2)$.

To estimate the distance between a state and a query the function *dist* : $S \times \mathbf{Cond} \to \mathbb{N}$ is introduced in Table 2. It makes use of the previously defined

function Δ to calculate the distance estimate of the comparison operators. Given a state S and a query q, *dist* yields an integer estimate of the distance from state S to a state S' satisfying the query q.

Note that we avoid handling the negation construction $\neg$, in the distance function (Table 2), by rewriting queries to a form without the use of negation using boolean rules, i.e. inverting operators and swapping conjuctions and disjunctions. When computing the distance for conjunctions we take the sum of the distance between two operands. Consequently, increasing the distance if there are many conjunctions with unsatisfied operands. For disjunctions the heuristic returns the distance to the most optimistic operand, i.e the operand with the lowest distance.

$$\begin{aligned} dist(s, e_1 \bowtie e_2) &= \Delta(eval(e_1, M, V), \bowtie, eval(e_2, M, V)) \\ &\quad \text{, where } s = (M, V) \text{ and } \bowtie \in \{=, <, \leq, >, \geq, \neq\} \\ dist(s, c_1 \wedge c_2) &= dist(s, c_1) + dist(s, c_2) \\ dist(s, c_1 \vee c_2) &= min(dist(s, c_1), dist(s, c_2)) \end{aligned}$$

Table 2: Formal *dist* Specification

We create a best first search algorithm by replacing line 5 in Algorithm 3 with $S = \arg\min_{S \in Q} dist(S, q)$. This heuristic operates under the assumption that similar states are likely to be a few firings away from each other. As we will see in Section 6.3, this assumption works for most of the queries used in the experiments. Nevertheless, it is possible to create a scenario where the heuristic estimate degrades.

This heuristic operates under the assumption that similar states are only a few firings away from each other. As Section 6.3 will show, this is true for many interesting queries, however it will likely be possible to create a PN where this infact leads to a longer search.

6.2 Over-Approximation Using Integer Programming

The techniques presented so far rely on searching through the state space. Now we will present a technique based on integer programming proposed by Esparza and Melzer in [7], which can be used to efficiently disprove reachability in some cases, by over-approximating the state space, hence avoiding state space explosion. The intuition behind this technique is that reachability can be ruled out if there does not exist any combination of transitions such that a target marking is reachable. This technique provides an over-approximation for ordinary PNs, but by discarding variables, pre- and post-conditions it can still be used for PNDVs.

Given a PN $N = (P, T, F)$ and markings $M_0, M' \in \mathcal{M}(N)$, if there is a sequence of transitions $M_0 \to \cdots \to M'$, such that M' is reachable from M_0, then there is a firing vector $\boldsymbol{\theta} : T \to \mathbb{N}$, such that, the state equation,

$$M_0(p) + \sum_{t \in T} (F(t, p) - F(p, t)) \cdot \boldsymbol{\theta}(t) = M'(p)$$

holds for all $p \in P$ [3]. Clearly, a solution to $\boldsymbol{\theta}(t)$ is the number of times t was fired in a sequence of transitions from M_0 to M'. Notice that a solution to the

state equation does not imply that M' is reachable, but if M' is reachable it implies the existence of a solution. Conversely, if there is no solution to the state equation, M' is not reachable from M_0. The state equation can be expressed as a matrix equation and solved using linear algebra, yet this does not ensure a positive integer solution. Esparza and Melzer proposed the usage of integer programming to solve the state equation in [7], ensuring $\boldsymbol{\theta}(t) \in \mathbb{N}$ for all $t \in T$, thus providing a more accurate approximation.

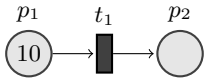


Fig. 7: A simple PN

To determine whether it is possible or not for the marked PN in Figure 7 to satisfy the query $c = (5 \leq p_1 \wedge p_1 \leq 7) \wedge (6 \leq p_2 \vee p_2 \leq 2)$, we can derive two systems of inequations

$$s_1 = \left\{ \begin{array}{r} 10 - 1 \cdot \boldsymbol{\theta}(t_1) \leq 7, \\ 10 - 1 \cdot \boldsymbol{\theta}(t_1) \geq 5, \\ 0 + 1 \cdot \boldsymbol{\theta}(t_1) \leq 2, \\ 0 + 1 \cdot \boldsymbol{\theta}(t_1) \geq 0 \end{array} \right\}, \quad s_2 = \left\{ \begin{array}{r} 10 - 1 \cdot \boldsymbol{\theta}(t_1) \leq 7, \\ 10 - 1 \cdot \boldsymbol{\theta}(t_1) \geq 5, \\ 0 + 1 \cdot \boldsymbol{\theta}(t_1) \leq \infty, \\ 0 + 1 \cdot \boldsymbol{\theta}(t_1) \geq 6 \end{array} \right\}$$

If s_1 and s_2 are given as input to an integer programming solver, such as lp_solve [1], it will tell us that no integer solution exists for neither s_1 nor s_2. From this we can conclude that a marking satisfying c is not reachable in Figure 7. If an integer solution to either s_1 or s_2 was found, an incremental trap-testing refinement of the state equation proposed in [7] can easily be applied, once a solution is found. Implementation and preliminary experimentation with this technique in PeTe gave promising results. For a complete presentation of how to derive these systems of inequations from a query see Appendix B.

6.3 Evaluation

To evaluate the algorithms presented earlier in this section, we have implemented PeTe, a Petri net modelling and verification tool for PNDVs. PeTe is written in Qt/C++, and comes with a GUI and many variations of the algorithms presented earlier. A screenshot of PeTe can be found in Figure 8 in Appendix C, sources and binaries can be obtained from [2].

The models used to evaluate the efficiency of the algorithms are ordinary bounded Petri nets from the SUMo model checking contest [4], to which PeTe was also submitted. The submission kit for this contest provided three models: FMS, Kanban and MAPK as well as ten satisfiable and ten non-satisfiable queries for each model. Note that these models are designed to be scalable in the number of tokens, such that when scaling certain places, verification becomes increasingly more computationally demanding.

The efficiency of best first search (BestFS), described in Section 6.1, was evaluated by comparing it with two naive approaches, breadth first search (BFS) and

random depth first search (RDFS). Due to the irregularity of RDFS, all results for this algorithm are the average of ten runs. All experiments were performed on an Intel Core 2 Duo, running Ubuntu 10.10, where memory usage was restricted to a maximum of $1GiB$ and the tool would terminate upon exceeding this limit.

Table 3 shows the average running times of the three search strategies. Each value is the average completion time of ten different queries on each of the three models. The suffix of each model name denotes the scaling factor[1]. Considering the table, it is clear that the naive search strategies were unable to verify queries for models with a higher scaling factor than the lowest one within the given memory constraints.

	Strategy		
	RDFS	**BFS**	**BestFS**
FMS 4	1.4379	1.516	0.002
FMS 10	Out of memory	Out of memory	0.074
FMS 20	Out of memory	Out of memory	0.111
FMS 50	Out of memory	Out of memory	0.646
Kanban 5	5.865	5.301	0.002
Kanban 10	Out of memory	Out of memory	0.057
Kanban 20	Out of memory	Out of memory	0.066
Kanban 50	Out of memory	Out of memory	0.155
MAPK 8	20.2094	27.119	0.004
MAPK 40	Out of memory	Out of memory	0.086
MAPK 80	Out of memory	Out of memory	0.182
MAPK 160	Out of memory	Out of memory	0.142

Table 3: PeTe running times on three different models with four different scaling factors using the given search strategies. Time was not measured when PeTe ran out of memory. All measurements are in seconds

From Table 3 it is clear that the BestFS is significantly faster than both BFS and RDFS. Moreover, the results show no considerable slowdown for BestFS as the scaling factor grows. We can conclude that BestFS is capable of handling larger models than the two naive approaches. For results for larger scaling factors see Appendix F. To further explore the limits of BestFS additional experiments

FMS		**Kanban**		**MAPK**	
Scale	#	Scale	#	Scale	#
50	0	100	1	320	0
100	71	200	4	640	7
200	0	500	6	1280	0
500	89	1000	11	2560	572

Table 4: Number of searches out of 1000 PeTe was unable to satisfy within the memory constraint

were performed by randomly generating 1000 satisfiable queries for each instance of the models. Table 4 presents the number of queries PeTe was unable to verify before terminating due to the memory constraint. Notice that not all scaling

[1] See Appendix D for details on how models are scaled.

factors make it more difficult to verify queries. We believe this is attributed to the structure and counting places in the models. Besides this anomaly, the number of unverifiable queries grows with the scaling factor. From Table 4 it can be concluded that it is possible to find queries that BestFS cannot verify, even for rather small instances of Kanban.

Finally, the SUMo model checking contest submission kit [4] also provided non-satisfiable queries. While BestFS provided no improvement over naive search for these queries, the over-approximation presented in Section 6.2 was able to disprove all of them, for any scaling of each model in less than one second.

7 Conclusion

We have proposed a new extension of Petri nets that permits more modelling convenience over classical Petri nets. This model is Turing complete and we have given a decidable subset with the same expressive power as Petri nets. We have demonstrated the modelling power of Petri nets with discrete variables and their applications by simulating bounded discrete timed-arc Petri nets, allowing one to decide reachability. Finally, we have developed heuristic algorithms for reachability analysis of Petri nets with discrete variables. We have confirmed their advantages over naive and randomized search through experimental results, showing that larger models may become tractable when heuristics are applied. Future development could include integrating over-approximation with best first search, hence enabling exclusion of unpromising branches. Alternatively, a framework for determining when over-approximation is exact could be developed.

References

1. lp_solve - linear and integer programming solver, `http://lpsolve.sourceforge.net/5.5/`
2. PeTe - modelling and verification tool for petri nets, `https://github.com/jopsen/PeTe`
3. Petri Nets: Properties, Analysis and Applications, vol. 77 (April 1989)
4. Sumo model checking contest (june 2011), `http://sumo.lip6.fr/Model_Checking_Contest.html`
5. Boucheneb, H., Gardey, G., Roux, O.H.: Tctl model checking of time petri nets. Journal of Logic and Computation 19(6), 1509–1540 (2009)
6. Byg, J., Jørgensen, K., Srba, J.: An efficient translation of timed-arc petri nets to networks of timed automata. In: Formal Methods and Software Engineering, LNCS, vol. 5885, pp. 698–716. Springer Berlin / Heidelberg (2009)
7. Esparza, J., Melzer, S.: Verification of safety properties using integer programming: Beyond the state equation. Formal Methods in System Design 16, 159–189 (2000)
8. David de Frutos Escrig, Valentín Valero Ruiz, O.M.A.: Decidability of properties of timed-arc petri nets. In: Proc. of the 21st international conference on Application and theory of petri nets. pp. 187–206. ICATPN'00, Springer-Verlag, Berlin, Heidelberg (2000)

9. Hanisch, H.M.: Analysis of place/transition nets with timed arcs and its application to batch process control. In: Ajmone Marsan, M. (ed.) Application and Theory of Petri Nets 1993, LNCS, vol. 691, pp. 282–299. Springer Berlin / Heidelberg (1993)
10. Jacobsen, L., Jacobsen, M., Møller, M., Srba, J.: A framework for relating timed transition systems and preserving tctl model checking. In: Computer Performance Engineering, LNCS, vol. 6342, pp. 83–98. Springer Berlin / Heidelberg (2010)
11. Jensen, K., Kristensen, L., Wells, L.: Coloured petri nets and cpn tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer (STTT) 9, 213–254 (2007)
12. Kosaraju, S.R.: Decidability of reachability in vector addition systems (preliminary version). In: Proc. of the 14th annual ACM symposium on Theory of computing. pp. 267–281. STOC '82, ACM, New York, NY, USA (1982)
13. Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to coloured petri nets. International Journal on Software Tools for Technology Transfer (STTT) 2, 98–132 (1998)
14. Mayr, E.W.: An algorithm for the general petri net reachability problem. In: Proc. of the 13th annual ACM symposium on Theory of computing. pp. 238–246. STOC '81, ACM, New York, NY, USA (1981)
15. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall PTR, Upper Saddle River, NJ, USA (1981)
16. Petri, C.: Kommunikation mit automaten. Tech. rep. (1966)
17. Ratzer, A., Wells, L., Lassen, H., Laursen, M., Qvortrup, J., Stissing, M., Westergaard, M., Christensen, S., Jensen, K.: Cpn tools for editing, simulating, and analysing coloured petri nets. In: van der Aalst, W., Best, E. (eds.) Applications and Theory of Petri Nets 2003, LNCS, vol. 2679, pp. 450–462. Springer Berlin / Heidelberg (2003)
18. Valentin Valero Ruiz, Fernando Cuartero Gomez, D.d.F.E.: On non-decidability of reachability for timed-arc petri nets. Eighth International Workshop on Petri Nets and Performance Models (PNPM '99) p. 188 (1999)

A Proof of Theorem 2

Theorem 2. *The labeled transition system (LTS) of a marked p-free PNDV (N_1, S_1) is isomorphic to the LTS of its translated marked PN (N_2, M_2), where $N_2 = \text{P-FREE-PNDV-TO-PN}(N_1)$ and any M_2 s.t. $S_1 \equiv M_2$.*

Proof. We prove isomorphism by showing that *(i)* the correspondence relation is a bijection, *(ii)* $S_1' \xrightarrow{t} S_1''$ implies $M_2' \rightarrow M_2''$, where $S_1'' \equiv M_2''$, and *(iii)* $M_2' \xrightarrow{t'} M_2''$ implies $S_1' \rightarrow S_1''$, where $S_1'' \equiv M_2''$.

Considering the correspondence relation in Definition 4 it is obvious that any state S_1' on N_1 has exactly one corresponding marking M_2' on N_2. Since the complement place is accounted for in the correspondence relation, any marking M_2' on N_2 for which a corresponding state S_1' on N_1 exists, S_1' is unique. Thus, the correspondence relation is a bijection between all states on N_1 and any marking on N_2 that has a corresponding state on N_1. Hence, we have shown (i).

We show (ii), by considering the transition t for which $S_1' = (M_1', V) \xrightarrow{t} S_1''$. By enabledness we know that valuation V satisfies $Pre(t)$. This valuation V must have been considered during the translation to PN, thus a transition t_V must exist in T_2, s.t. $M_2' \xrightarrow{t_V} M_2''$, where $S_1'' \equiv M_2''$.

(iii) follows from the consideration that in $M_2' \xrightarrow{t'} M_2''$ the transition t' must be of the form $t' = t_V$, for some valuation V, s.t. $S_1' = (M_1', V)$, otherwise M_2' and S_1' would not correspond. From the translation we know that $t \in T_1$, where $V \models Pre(t)$, must exist s.t. $S_1' \xrightarrow{t} S_1''$ and $S_1'' \equiv M_1''$. Thus, we have shown (i), (ii) and (iii), proving isomorphism. □

B Over-Approximation Using Integer Programming

In this section elaborate on the derivation of inequations from a reachability query. First we define a reduced condition language $\mathbf{Cond_r}$ for expressing reachability queries,

$$\mathbf{Cond_r} ::= \mathbf{p} \bowtie \mathbf{z} \mid \mathbf{c_r} \vee \mathbf{c_r} \mid \mathbf{c_r} \wedge \mathbf{c_r}$$

, where $\bowtie \in \{=, <, \leq, >, \geq, \neq\}$, $z \in \mathbb{Z}$ and $p \in P$. For simplicity negation is not included, however it can be easily achieved by rewrite the formula using the rules for boolean logic.

A marking constraint $y \in CON = P \rightarrow \{[a, b] \mid a \in \mathbb{N}, b \in \mathbb{N} \cup \{\infty\}\}$ is a mapping from places to intervals. For convenience we let $y = \{p \mapsto [z_1; z_2]\}$, where $p \in P$ and $z_1, z_2 \in \mathbb{Z}$, denote a constraint $y \in CON$, such that $y(p) = [z_1, z_2]$ and $y(p') = [0, \infty]$ for all $p' \in P \setminus \{p\}$. In the end each marking constraint will be translated into a system of inequations. To deduce a set of marking constraints from a query, we introduce an auxiliary function $cons : \mathbf{Cond_r} \rightarrow \mathbf{2}^{CON}$, as defined in Table 5.

The function *cons* is defined such that a marking M satifies a query c_r, if and only if there exists a constraint $y \in cons(c_r)$ where $M(p) \in y(p)$ for all $p \in P$. This property enables us to exploit the expressive power of integer

$$
\begin{array}{lll}
cons(c1 \wedge c2) = \{combine(y1, y2) \mid y1 \in cons(c1), y2 \in cons(c2)\} & & \\
\quad \text{where } combine(y1, y2)(p) = y1(p) \cap y2(p), \text{ for all } p \in P & & \\
cons(c1 \vee c2) = cons(c1) \cup cons(c2) & & \\
cons(p = z) = \{\{p \mapsto [k; z]\}\} & cons(p \neq z) = \{\{p \mapsto [0; z-1]\}, \{p \mapsto [z+1; \infty]\}\} & \\
cons(p \leq z) = \{\{p \mapsto [0; z]\}\} & cons(p \geq z) = \{\{p \mapsto [z; \infty]\}\} & \\
cons(p < z) = \{\{p \mapsto [0; z-1]\}\} & cons(p > z) = \{\{p \mapsto [z+1; \infty]\}\} &
\end{array}
$$

Table 5: Auxiliary function for extracting constraints

programming to create a system of inequations for each $y \in cons(c_r)$ and if none of these systems have a positive integer solution, we can conclude that a marking satifying c_r is not reachable.

Algorithm 4 Over-approximation Using Integer Programming

```
1: function CAN-DISPROVE-REACHABILITY(q, M_0, N)
2:     (P, T, F) = N
3:     for all y ∈ cons(q) do
4:         sys = ∅                          ▷ Let sys be an empty system of inequations
5:         for all p ∈ P do
6:             [min, max] = y(p)
7:             sys = sys ∪ M_0(p) + Σ_{t∈T} (F(t,p) − F(p,t)) · θ(t) ≥ min
8:             sys = sys ∪ M_0(p) + Σ_{t∈T} (F(t,p) − F(p,t)) · θ(t) ≤ max
9:         end for
10:        if there is an integer solution to θ satisfying constraints in sys then
11:            return "Conclude marking satisfying q might be reachable"
12:        end if
13:    end for
14:    return "Conclude marking satisfying q is not reachable"
15: end function
```

Algorithm 4 shows how the marking constraints derived with *cons* can be used to construct the systems of inequations that is sufficient to prove a query not reachable.

C PeTe

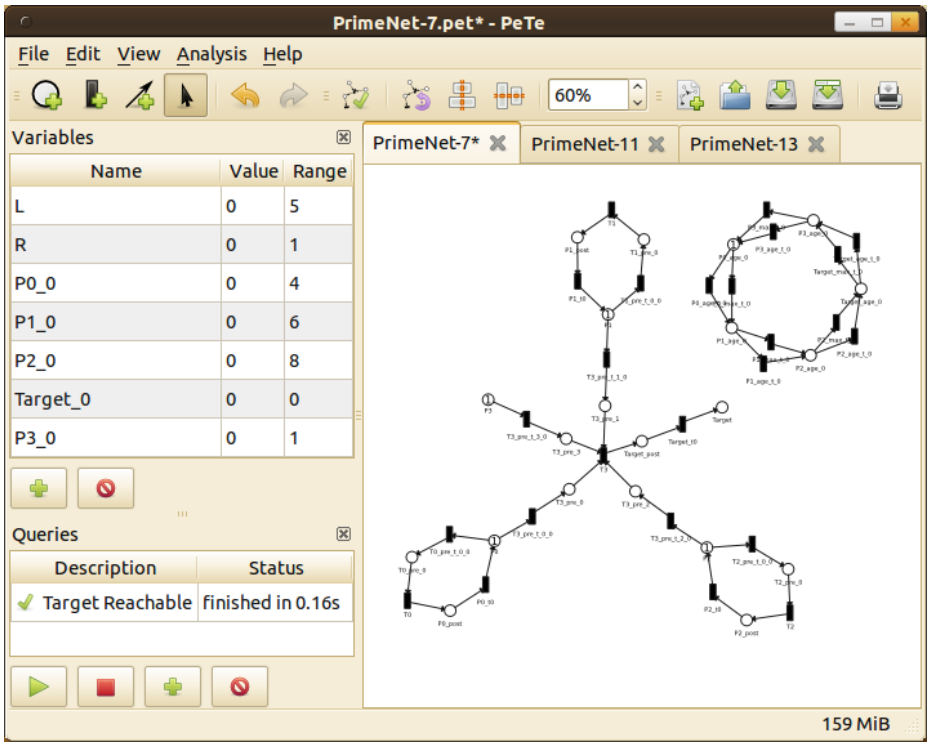


Fig. 8: A screenshot of PeTe, demonstating the model editor, and the side bars for queries and variables

D Models Used for Experiments

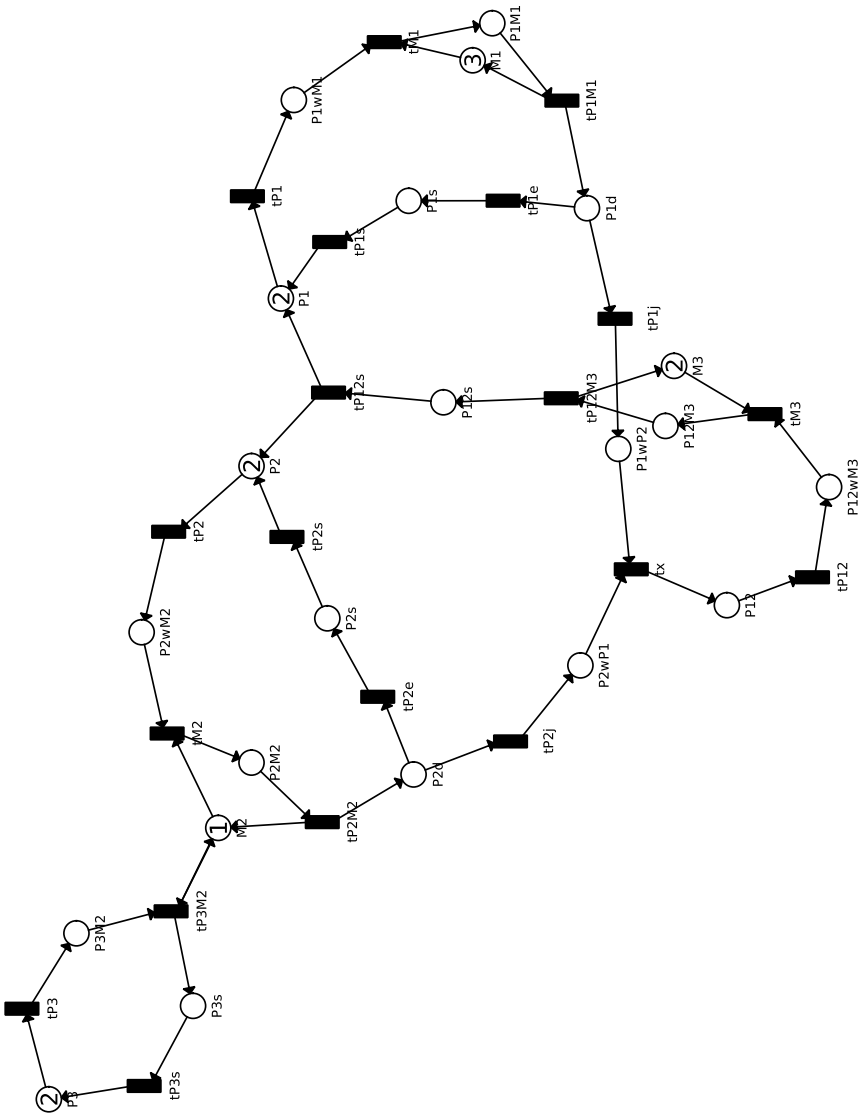


Fig. 9: FMS model used for experiments. FMS is scaled by increasing the number of tokens in places $P1$, $P2$ and $P3$

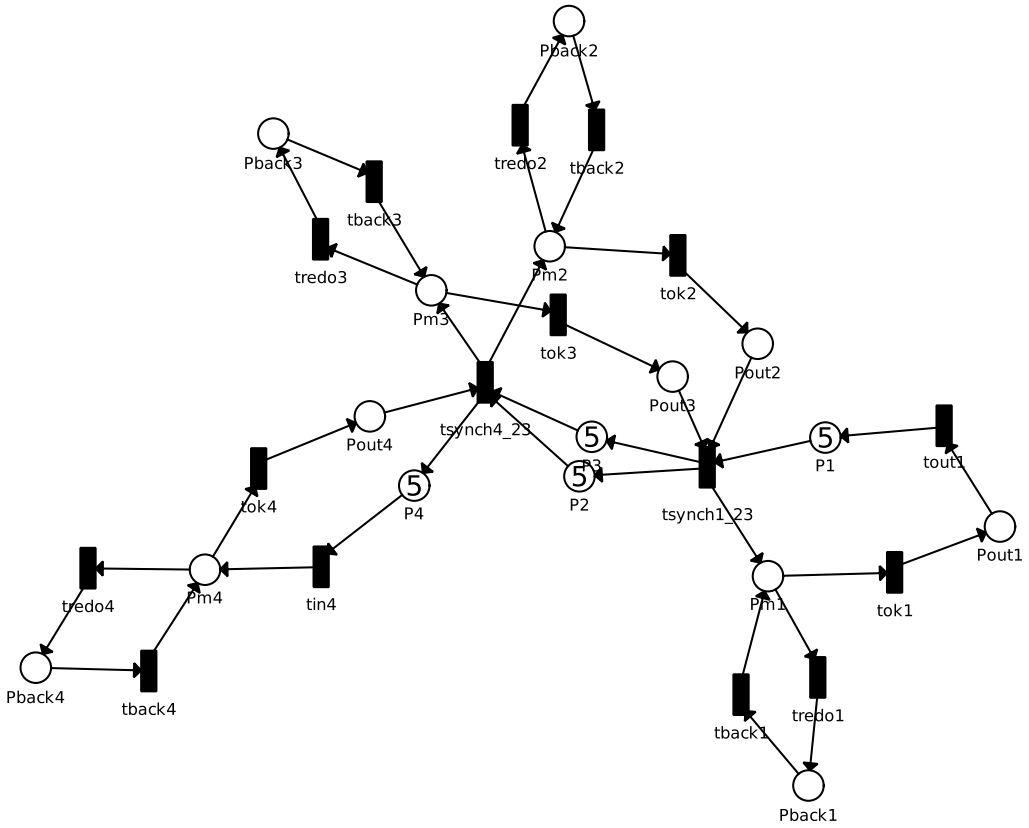


Fig. 10: Kanban model used for experiments. Kanban is scaled by increasing the number of tokens in places *P1, P2, P3* and *P4*

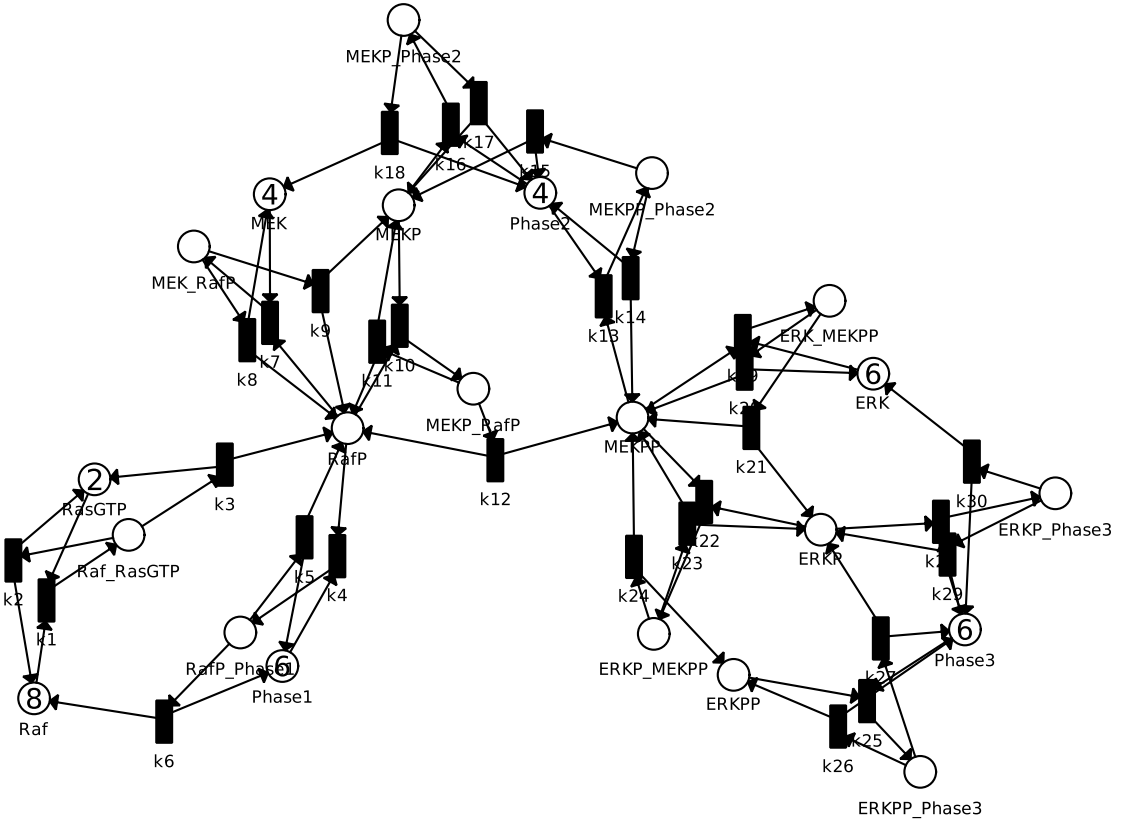


Fig. 11: MAPK model used for experiments. MAPK is scaled by increasing the number of tokens in places *Phase1, Phase2, Phase3, MEK, ERK, Raf* and *RasGTP*

E Running Times for FMS and MAPK

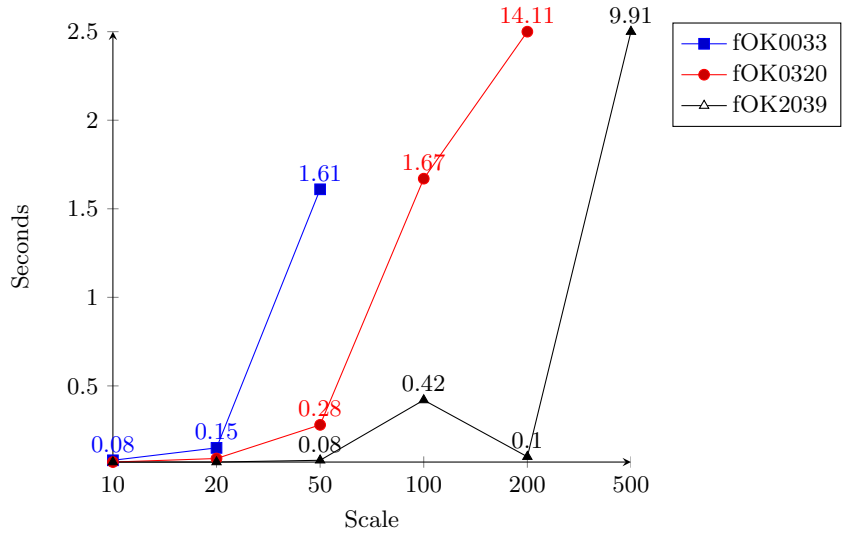


Fig. 12: Running Times for three queries on FMS. Each line represents the running time for a selected query. These queries were provided by the SUMo Model Checking Contest submission kit [4]. All measurements are in seconds

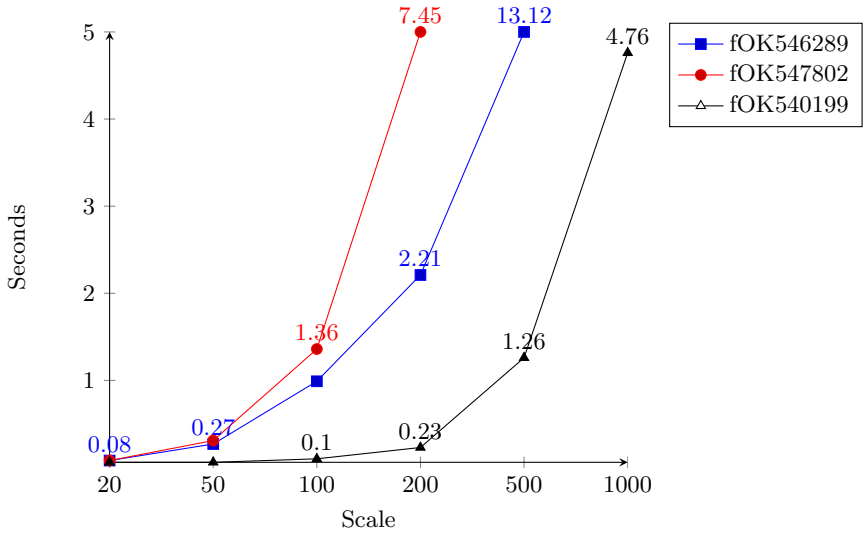


Fig. 13: Running times for three queries on Kanban. Each line represents the running time for a selected query. These queries were provided by the SUMo Model Checking Contest submission kit [4]. All measurements are in seconds

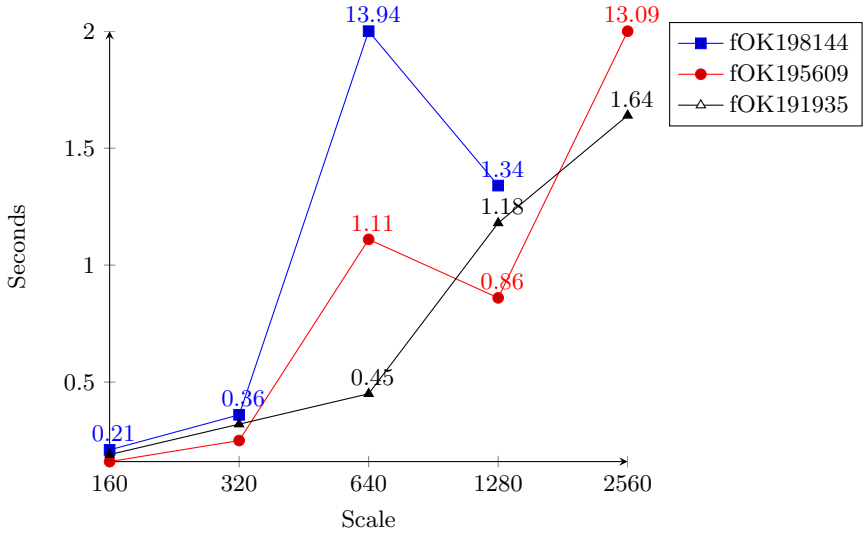


Fig. 14: Running times for three queries on MAPK. Each line represents the running time for a selected query. These queries were provided by the SUMo Model Checking Contest submission kit [4]. All measurements are in seconds

F Experimental Results

Model scale	Query	Running Time	Expanded States	Explored States	Path Length
FMS 4	fOK0320	0	87	342	30
FMS 4	fOK0033	0	176	480	44
FMS 4	fOK2039	0	34	165	34
FMS 10	fOK0320	0.07	324	1137	66
FMS 10	fOK0033	0.08	1751	4704	141
FMS 10	fOK2039	0.07	295	1568	85
FMS 20	fOK0320	0.09	1672	5499	126
FMS 20	fOK0033	0.15	8867	23587	371
FMS 20	fOK2039	0.07	170	917	170
FMS 50	fOK0320	0.28	18787	57939	306
FMS 50	fOK0033	1.61	154328	381144	1261
FMS 50	fOK2039	0.08	425	2327	425
FMS 100	fOK0320	1.67	135909	411537	606
FMS 100	fOK0033	Out of memory	Out of memory	Out of memory	Out of memory
FMS 100	fOK2039	0.42	20950	110468	850
FMS 200	fOK0320	14.11	1042612	3141639	1206
FMS 200	fOK0033	Out of memory	Out of memory	Out of memory	Out of memory
FMS 200	fOK2039	0.1	1700	9377	1700
FMS 500	fOK0320	Out of memory	Out of memory	Out of memory	Out of memory
FMS 500	fOK0033	Out of memory	Out of memory	Out of memory	Out of memory
FMS 500	fOK2039	9.91	504750	2652468	4250

Table 6: Table representing experimental data performed by PeTe on three queries from the FMS model supplied by the submission kit from [4]. Running time is the time in seconds PeTe took to find a state satisfying the query. Expanded states are the number of states from which all child states have been visited. Explored states is the number of states visited in total. Path length is the length of the path from the initial state, to the state satisfying the query

Model scale	Query	Running Time	Expanded States	Explored States	Path Length
Kanban 5	fOK546289	0	267	1401	48
Kanban 5	fOK540199	0	60	428	60
Kanban 5	fOK547802	0	285	1629	48
Kanban 10	fOK546289	0.06	784	4964	96
Kanban 10	fOK540199	0.05	132	937	132
Kanban 10	fOK547802	0.06	560	3606	122
Kanban 20	fOK546289	0.08	2006	13558	192
Kanban 20	fOK540199	0.06	287	2145	284
Kanban 20	fOK547802	0.08	1362	8839	276
Kanban 50	fOK546289	0.27	13870	85287	480
Kanban 50	fOK540199	0.06	780	6580	780
Kanban 50	fOK547802	0.31	17464	106279	498
Kanban 100	fOK546289	0.99	52150	319537	960
Kanban 100	fOK540199	0.1	1960	17390	1960
Kanban 100	fOK547802	1.36	80744	479694	998
Kanban 200	fOK546289	2.21	92402	719878	1920
Kanban 200	fOK540199	0.23	8531	77169	8492
Kanban 200	fOK547802	7.45	372600	2268247	7836
Kanban 500	fOK546289	13.12	531062	4224118	4800
Kanban 500	fOK540199	1.26	45311	426849	45212
Kanban 500	fOK547802	Out of memory	Out of memory	Out of memory	Out of memory
Kanban 1000	fOK546289	Out of memory	Out of memory	Out of memory	Out of memory
Kanban 1000	fOK540199	4.76	170611	1633649	170412
Kanban 1000	fOK547802	Out of memory	Out of memory	Out of memory	Out of memory

Table 7: Table representing experimental data performed by PeTe on three queries from the Kanban model supplied by the submission kit from [4]. Running time is the time in seconds PeTe took to find a state satisfying the query. Expanded states are the number of states from which all child states have been visited. Explored states is the number of states visited in total. Path length is the length of the path from the initial state, to the state satisfying the query

Model scale	Query	Running Time	Expanded States	Explored States	Path Length
MAPK 8	fOK195609	0	127	972	62
MAPK 8	fOK191935	0.01	133	1336	69
MAPK 8	fOK198144	0.01	155	1346	90
MAPK 40	fOK195609	0.09	748	6942	206
MAPK 40	fOK191935	0.1	929	9001	261
MAPK 40	fOK198144	0.1	834	10756	272
MAPK 80	fOK195609	0.12	1620	14268	710
MAPK 80	fOK191935	0.11	1281	13958	590
MAPK 80	fOK198144	0.35	7553	83499	4644
MAPK 160	fOK195609	0.16	3088	28842	806
MAPK 160	fOK191935	0.19	3809	37321	1026
MAPK 160	fOK198144	0.21	3406	44389	1080
MAPK 320	fOK195609	0.25	6208	58042	1606
MAPK 320	fOK191935	0.32	7649	75081	2046
MAPK 320	fOK198144	0.36	6836	89249	2160
MAPK 640	fOK195609	1.11	35360	276978	16810
MAPK 640	fOK191935	0.45	10206	111748	4650
MAPK 640	fOK198144	13.94	351187	3839955	225900
MAPK 1280	fOK195609	0.86	24928	233242	6406
MAPK 1280	fOK191935	1.18	30689	301641	8166
MAPK 1280	fOK198144	1.34	27416	358409	8640
MAPK 2560	fOK195609	13.09	448640	3335298	220810
MAPK 2560	fOK191935	1.64	40806	447028	18570
MAPK 2560	fOK198144	Out of memory	Out of memory	Out of memory	Out of memory

Table 8: Table representing experimental data performed by PeTe on three queries from the MAPK model supplied by the submission kit from [4]. Running time is the time in seconds PeTe took to find a state satisfying the query. Expanded states are the number of states from which all child states have been visited. Explored states is the number of states visited in total. Path length is the length of the path from the initial state, to the state satisfying the query