Efficient Unfolding and Approximation of Colored Petri Nets with Inhibitor Arcs

Andreas H. Klostergaard

June 8, 2018

Abstract

We define colored Petri nets with inhibitor arcs, and present an unfolding method, which allows us to unfold these to Petri nets with inhibitor arcs. We also present an overapproximation algorithm, which can answer a subset of queries performed on colored Petri nets, without unfolding the net. This algorithm offers faster verification of colored Petri nets, for the queries we are able to answer. In nets like *BART-COL* from the MCC'2017 competition, which has never received any answers by any tools in the competition, we are able to answer 62 out of 128 queries, using this algorithm. We implement both the overapproximation algorithm and the unfolding method in the *verifypn* tool. Using the nets from the MCC'2017 competition as test set, we compare the unfolding implementation to the one in the tool *MCC*, where we are on average 30% slower at unfolding, but are faster in total run time in every net except one.

1 Introduction

In the modern world where systems grow in complexity, we find the need for tools to verify these systems, to prove that they work as intended. For this purpose we invent different models, as to be able to represent these systems in a way, in which we can do verification to see if a property is satisfied in the modelled system. One such model is Petri nets first stipulated by C.A. Petri in 1962 [14]. Since then this model has received numerous extensions, where some of these include timed arcs [15], inhibitor arcs [8], colors [10], and more. Some of these extensions were made in order to increase the expressiveness of the model, such as inhibitor arcs, which make the model Turing complete, while other extensions serve as higher abstractions of the model. Colored Petri nets is such an extension.

In this thesis, we have a closer look at colored Petri nets with inhibitor arcs. In 1981 K. Jensen proposed the extension, colored Petri nets in [10]. This extension made it easier to model complex system, since a single place can now hold different colored tokens, i.e. tokens with different values, and arcs can selectively pick which color the tokens must have, in order for it to be consumed or produced. The arcs can not only select the colors, they can also contain variables. The transitions are able to define guards, which can then put restrictions on which colors can bind to the variables of the arcs. Since this model is still a higher abstraction of Petri nets, it does not increase the expressiveness.

Later in 1988 J. Billington took Jensen's colored Petri nets as a starting point, and extended it with capacity and inhibitor functions [1]. His definition of colored Petri nets varies from the one which appeared in Jensen's work, but remains compatible. We use the initial idea of the inhibitor arcs from Billington, and adapt it to the colored Petri nets defined in [9].

In addition to defining inhibitor arcs for Jensen's colored Petri nets, we also extend the unfolding of colored Petri nets defined in [11], with the addition of inhibitor arcs.

The unfolding of colored Petri nets often causes an explosion in net size, which can lead to very large state spaces. In these cases, unfolding and verification of the unfolded net can be a very costly operation. Because of cases like these we formulate an algorithm, which overapproximates the colored Petri net, by stripping away the colors. This allows us to still verify some aspects of the net, while leaving others with inconclusive answers. An example of a net that is too costly to unfold within a reasonable amount of time and memory is the net *BART-COL* from the competition MCC'2017 [13]. With the overapproximation algorithm we can now answer a subset of queries on nets like these. In the category *ReachabilityCardinality* of the MCC'2017 competition, we answered 42.9% of the queries, while in the *Reachability-Fireability* category, we are only able to answer 6% of the queries.

We implement the representation and unfolding of colored Petri nets in the tool *verifypn* [2]. This implementation does not include inhibitor arcs for colored Petri nets. The reason behind this is due to the standard for representing Petri nets, and High-level Petri nets, named PNML [16], does not support inhibitor arcs for colored nets. We also implement the overapproximation algorithm in the same tool.

Lastly we compare how many queries can be answered using the overapproximation algorithm, compared to unfolding and then verifying. We also compare the speed of the overapproximation algorithm, to the speed of unfolding and then verifying. For this we use the nets and queries from the MCC'2017 competition [13].

The unfolding speed will also be compared to an existing solution called MCC[1], which only does unfolding, while requiring another tool for verification. On average we are 29.89% slower in unfolding the nets, but we are faster in total run time in all nets except one.

2 Preliminaries

In this section we describe some of the preliminaries that we need in order to understand how colored Petri nets work. To start off, we first introduce Petri nets with inhibitor arcs, as this is the model that we unfold to. After this we describe the concept of multisets, and the operators that we use for the rest of this thesis.

2.1 Definition of Petri Nets with Inhibitor Arcs

In this subsection we define Petri nets with inhibitor arcs. The notion of Petri nets was first stipulated in [14], which was later extended with inhibitor arcs in [8].

Definition 1. (PN)
A **Petri net** is a six tuple $\mathbf{PN} = (P, T, F, W, I, W^I)$ where:

1. P is a finite set of **places**,
2. T is a finite set of **transitions**,
3. $F \subseteq (P \times T) \cup (T \times P)$ is the set of **arcs**,
4. $W : F \to \mathbb{N}$ is the **weight** function,
5. $I \subseteq P \times T$ is the set of **inhibitor arc**, and
6. $W^I : I \to \mathbb{N}$ is the **inhibitor weight** function.

An example of a Petri net can be found in Figure 1, which models a parallel production line. Before going in depth with this example, we have to define what a marking is, and how we transition between markings.

A marking can then be defined as following:

Definition 2. (PN Marking)
A **marking** M in a **PN** is defined as a function $M : P \to \mathbb{N}^0$, i.e. returning the number of tokens in a given place.

[1] `https://github.com/dalzilio/mcc`

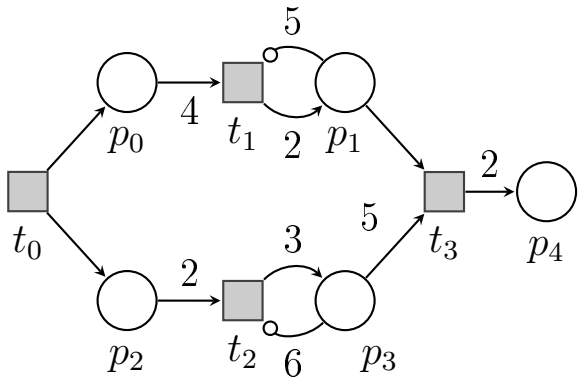


Figure 1: A Petri net modeling a parallel production line.

Given a marking, we can then define the concept of enabledness of a transition in a given marking.

Definition 3. (PN Enabledness)
A **transition** $t \in T$ is said to be enabled in marking M iff the following properties are satisfied:

1. $\forall (p,t) \in F : M(p) \geq W((p,t))$
2. $\forall (p,t) \in I : M(p) < W^I((p,t))$

I.e. that every place p with an arc going to transition t must have at least the amount of tokens, as the weight of the arc, and no place has more tokens than the weight of their inhibitor arcs to the transition.

The concept of enabledness is important, as it defines when a transition is able to fire, i.e. transitioning from one marking into another. The firing of a transition is defined as following:

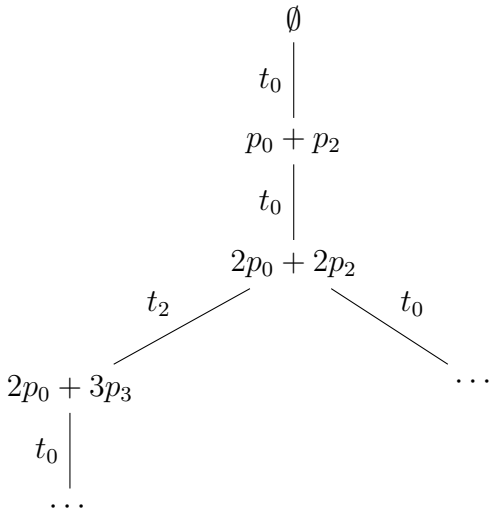
Definition 4. (PN Transition firing)
When a transition $t \in T$ is enabled in a marking M_1, it may fire, changing the marking M_1 to marking M_2, which is defined by:

$$\forall p \in P : M_2(p) = M_1(p) - W((p,t)) + W((t,p))$$

M_2 is said to be directly reachable from M_1 by the firing of transition t, defined by:

$$M_1 \xrightarrow{t} M_2.$$

The firing of a transition is what allows a Petri net to change states, thus allowing us to show the behaviour of a net.



Looking at Figure 1, we see that the current marking is an empty marking. This is due to the nature of this net. On the right we see a fragment of the LTS for this net. This net models a parallel production line, where the transition t_0 represents a supplier delivering materials to the start of the two production lines starting at place p_0 and p_2 respectively. We see that each line can move onto the next stage in production, independent of each other. Looking at the top production beginning at place p_0, we see that in order to fire transition t_1, we find that we need at least four tokens in order to move on in the line, and that we produce two tokens to place p_1, which is the next stage in production. We also have a restriction on transition t_1 in form of the inhibitor arc from place p_1, which holds production, if we have five or more tokens in place p_1, acting as a production limit, making sure that both production lines do not get out of sync. In the lower production line starting at place p_2, we see that this line is similar to the top one, except that this line consumes two tokens when firing transition t_2, and produces three tokens in place p_3. This production line is limited to six tokens in place p_3, as seen by the inhibitor arc. This production ends by collecting the results from both lines, by firing transition t_3, which consumes one token from place p_1 and five tokens from place p_3. Two tokens are then placed in place p_4, which represents the end product.

2.2 Multisets

In this subsection we introduce multisets, the notations, and operators used in this thesis. Multisets are used later when we introduce the colors of colored Petri nets.

Definition 5. (Multisets)
A **multiset** b, over a non-empty and *finite* set A, is a function from A to $\mathbb{N}^0$, i.e. $b \in A \to \mathbb{N}^0$. If $a \in A$ then $b(a)$ is the number of occurrences of a in the multi-set b.

We commonly represent a multiset b by a formal sum:

$$\sum_{a \in A} b(a)\,'(a).$$

The set of all multisets over a set A is denoted by A_{MS}. The empty multi-set is a multiset where all the coefficients are zero, and is denoted by $\emptyset$.

We also allow for infinite multisets, e.g. an infinite multiset over A is the function $A \to (\mathbb{N}^0 \cup \infty)$, and the set of all infinite multisets is denoted by A_{MS_∞}. Thus $A_{MS} \subset A_{MS_\infty}$.

Consider the set $A = \{x, y, z\}$, the multisets $2'(x)$, $1'(x) + 5'(y) + 2'(z)$, and $\emptyset$ are all members of A_{MS}, but where $\infty'(x) + 1'(y)$ is not a member of A_{MS}, although it is a member of A_{MS_∞}. I.e. the multiset containing two occurrences of x, and the multi-set containing one occurrence of x, five occurrences of y, and two occurrences of z, and the empty multi-set are all members of A_{MS}.

Definition 6. (Multiset operations)
Suppose A is a set, $b_1, b_2 \in A_{MS_\infty}$, $c \in A$, and $n \in \mathbb{N}$:

$c \in b_1$ iff $b_1(c) > 0$	(membership)
$b_1 \leq b_2$ iff $\forall a \in A : b_1(a) \leq b_2(a)$	(inclusion)
$b_1 = b_2$ iff $b_1 \leq b_2$ and $b_2 \leq b_1$	(equality)
$b_1 \uplus b_2 = \sum_{a \in A}(b_1(a) + b_2(a))\,'(a)$	(summation)
$n * b_1 = \sum_{a \in A}(n * b_1(a))\,'(a)$	(scalar-multiplication)
$\lvert b_1 \rvert = \sum_{a \in A} b_1(a)$	(cardinality)

When $b_1, b_2 \in A_{MS}$ and $b_2 \leq b_1$, subtraction is defined:

$b_1 \setminus b_2 = \sum_{a \in A}(b_1(a) - b_2(a))\,'(a)$	(subtraction)

Multisets are defined in more depth in [9, 1].

3 Colored Petri Nets with Inhibitor Arcs

In this section we describe what colored Petri nets are, how we query them, and how we can unfold them. First we describe what colors are, and how they are defined. Next we define three different type expressions, namely color-, guard-, and arc expressions. These expressions are used to inscribe either arcs or transitions. After this, we define colored Petri nets with inhibitor arcs, using the concepts defined previously.

We also list the CTL syntax used for queries on the colored Petri nets, while going into detail with the atomic propositions used.

Lastly we define how we unfold the colored Petri nets with inhibitor arcs into uncolored Petri nets with inhibitor arcs. We also discuss how we are able to transform atomic propositions of CTL queries to match the unfolded nets.

3.1 Colors

The color of a token in a colored Petri net is a value of the token. The set of all colors is defined as $\mathbb{C}$. This color value must be contained in one of the color sets defined for the net, where set of all color sets is defined as $\Sigma \subseteq 2^{\mathbb{C}}$.

In the PNML standard used in the MCC competition [13], different types of colors are defined. The different types of colors are *neutral colors*, *non-ordered colors*, and *ordered colors* [4]. Color types can also be combined in products, which are also a color type, e.g. assume we have colors $a \in A$ and $b \in B$, then $(a, b) \in A \times B$, which is also a color.

These types are defined in the PNML as the data types listed below. First we define three basic color types, and lastly we define a compound color type called product colors.

Dots. This type is the neutral color, equalling the tokens in a regular Petri net. The dot color type is as such always defined as $dot = \{\bullet\}$.

Finite enumerations. This type defines a finite set of user defined constants with no order. E.g. if we wanted to define a Boolean-like color type, where the constant values have no relation, such as the type $\{true, false\}$. This color type can then be defined as any finite set.

Cyclic enumerations. This type is an extension of the finite enumeration. It extends the finite enumerations by adding an order, and in turn a successor and predecessor function for each constant in the set, named *Succ* and *Pred*, respectively. Such as a set of age groups, e.g.

$$(baby, teenager, youngAdult, grownUp, pensioner)$$

where each color is preceded by another, and the successor of a *pensioner* is a *baby*.

In order to be able to evaluate orders using operators such as $<$, we assume the order is as the set is defined, e.g. in the example above the smallest element is *baby*, i.e. $baby < teenager$, and the largest element is *pensioner*, i.e. $grownUp < pensioner$, but note that $pensioner \not< baby$. This is an

implication of the order and is not necessarily related to the successor or predecessor functions.

As this color type requires a finite set, a successor function, and a predecessor function, cyclic enumerations are defined as a tuple $(A, Succ, Pred)$, where A is a finite set.

Range of integers. This type is used to represent numbered colors, which gives a natural ordering of the colors, and thereby specializes cyclic enumerations, by only allowing integer colors. E.g. the integer range $4 \ldots 9$, corresponding to the color type $[4, 5, 6, 7, 8, 9]$.

Since this is a specialization of cyclic enumerations, it is still defined as a tuple $([a, b], Succ, Pred)$, where a and b are the lower and upper bound, respectively. In this specialization $Succ$ and $Pred$ is always defined as:

$$Succ(x) = \begin{cases} x+1, & \text{if } a \leq x < b \\ a, & \text{if } x = b \end{cases} \qquad Pred(x) = \begin{cases} x-1, & \text{if } a < x \leq b \\ b, & \text{if } x = a \end{cases}$$

Product colors. This type is created be combining other colors. These colors are also called domains, and consist of the Cartesian product of all the constituent colors, i.e. $T_1 \times \cdots \times T_n$ is a domain if $T_1, \cdots, T_n \in \Sigma$, and then $T_1 \times \cdots \times T_n \in \Sigma$ such that if $c_1 \in T_1, \cdots, c_n \in T_n$ then $(c_1, \cdots, c_n) \in T_1 \times \cdots \times T_n$.

3.1.1 Variables, Types, and Bindings

In the following sections, we define arc- and guard expressions. In these expressions we need to define variables and types. From this point on the type of a color will no longer refer to whether it is ordered, non-ordered, or neutral, but rather the color set a given color belongs to.

The expressions are described in more detail in the following subsections. They have no side-effects and the variables, denoted by v, in them are bound to values, instead of assigning values to them, like in functional programming languages. The set of all variables is denoted by Var. These variables have a type, meaning that the value bound to them must be one of the colors in the color type. The type of a variable is a function $Type : Var \to \Sigma$.

As mentioned, variables need bindings. The set of all bindings is denoted by $\mathbb{B}$, and a binding $b \in \mathbb{B}$ is a function $b : Var \to \mathbb{C}$, such that $b(v) \in Type(v)$. The binding of a set of variables $Var = \{v_1, v_2, \ldots, v_n\}$ is denoted by $b = \langle v_1 = c_1, v_2 = c_2, \ldots, v_n = c_n \rangle$, and it is required that $c_i \in Type(v_i)$ for each variable $v_i \in Var$.

3.2 Color Expressions

The color expression of $\tau \in \mathbb{T}$, where $\mathbb{T}$ is the set of all arc expression, is defined as:

$$\tau ::= \sigma \mid (\tau, \cdots, \tau)$$
$$\sigma ::= \bullet \mid constant \mid var \mid \sigma{+}{+} \mid \sigma{-}{-}$$

Where $\bullet$ is the neutral color type named dot, *constant* is one of the allowed colors of the associated place (see Section 3.5), $var \in Var$ is a variable which represents one of the allowed colors of the associated place, $(\tau, \cdots, \tau)$ is a product color defining the color consisting of the individual colors defined in the expression, and $\sigma{+}{+}$ and $\sigma{-}{-}$ are the successor and predecessor, respectively, of the color represented by σ. The successor and predecessor are only available when the color type allows it, since not all PNML data types support them (see Section 3.1).

3.2.1 Semantics

To evaluate τ expressions in a given binding we define the function: $[\![]\!] : \mathbb{T} \times \mathbb{B} \to \mathbb{C}$. This function is used to evaluate a color sub-expression, and can thusly be applied recursively. The function takes a color sub-expression and a binding, and returns a color.

Definition 7. (τ-semantics)

$[\![(\bullet)\langle b \rangle]\!] = \bullet$	(neutral-color)
$[\![(constant)\langle b \rangle]\!] = constant$	(constant)
$[\![(var)\langle \ldots, var = c, \ldots \rangle]\!] = c$	(variable)
$[\![(\sigma{+}{+})\langle b \rangle]\!] = Succ([\![(\sigma)\langle b \rangle]\!])$	(successor)
$[\![(\sigma{-}{-})\langle b \rangle]\!] = Pred([\![(\sigma)\langle b \rangle]\!])$	(predecessor)
$[\![((\tau, \cdots, \tau))\langle b \rangle]\!] = ([\![(\tau)\langle b \rangle]\!], \cdots, [\![(\tau)\langle b \rangle]\!])$	(product)

An example of a color expression could be $(x{+}{+}, y)$, which denotes the product color, consisting of the successor of variable x, and the variable y. We can access the variables in the expression by the function $Var((x{+}{+}, y)) = \{x, y\}$. Then to find the type of both x and y, we can use the function $Type(x) = \{1, 2, 3\}$ and $Type(y) = \{4, 5, 6\}$. With this knowledge we can now evaluate the expression under some binding, such as $[\![(x{+}{+}, y)\langle x = 2, y = 6 \rangle]\!] = (3, 6)$.

3.3 Guard Expressions

The set of all guard expressions is defined as Γ, and a guard expression follows the following syntax:

$$\gamma ::= true \mid false \mid \neg\gamma \mid \gamma_1 \vee \gamma_2 \mid \gamma_1 \wedge \gamma_2 \mid \gamma_1 \rightarrow \gamma_2 \mid \gamma_1 \leftrightarrow \gamma_2 \mid \gamma_1 \operatorname{xor} \gamma_2 \mid \tau_1 \bowtie \tau_2$$

Where $\bowtie$ is one of the allowed comparison operators, i.e. $<$, $\leq$, $>$, $\geq$, $=$, and $\neq$.

3.3.1 Semantics

All guard expressions with a given binding evaluate to a Boolean value of either true or false. To evaluate a guard expression we define the function $[\![]\!] : \Gamma \times \mathbb{B} \rightarrow \{true, false\}$. This evaluation is later used for determining whether a given binding satisfies the guard expression. The function is defined as following:

Definition 8. (Guard semantics)

$[\![(\neg\gamma)\langle b\rangle]\!] = \neg[\![(\gamma)\langle b\rangle]\!]$	(negation)
$[\![(\gamma_1 \vee \gamma_2)\langle b\rangle]\!] = [\![(\gamma_1)\langle b\rangle]\!] \vee [\![(\gamma_2)\langle b\rangle]\!]$	(or)
$[\![(\gamma_1 \wedge \gamma_2)\langle b\rangle]\!] = [\![(\gamma_1)\langle b\rangle]\!] \wedge [\![(\gamma_2)\langle b\rangle]\!]$	(and)
$[\![(\gamma_1 \rightarrow \gamma_2)\langle b\rangle]\!] = [\![(\gamma_1)\langle b\rangle]\!] \rightarrow [\![(\gamma_2)\langle b\rangle]\!]$	(implication)
$[\![(\gamma_1 \leftrightarrow \gamma_2)\langle b\rangle]\!] = [\![(\gamma_1)\langle b\rangle]\!] \leftrightarrow [\![(\gamma_2)\langle b\rangle]\!]$	(bi-implication)
$[\![(\gamma_1 \operatorname{xor} \gamma_2)\langle b\rangle]\!] = [\![(\gamma_1)\langle b\rangle]\!] \operatorname{xor} [\![(\gamma_2)\langle b\rangle]\!]$	(xor)
$[\![(\tau_1 \bowtie \tau_2)\langle b\rangle]\!] = [\![(\tau_1)\langle b\rangle]\!] \bowtie [\![(\tau_2)\langle b\rangle]\!]$	(comparison)

An example of a guard expression g could be $(a < 4 \wedge a + + > b)$, where $Var(g) = \{a, b\}$. If we have that $Type(a) = Type(b) = \{1, 2, 3, 4\}$, then we can evaluate this expression with a binding such as $[\![g\langle a = 3, b = 3\rangle]\!] = true$.

3.4 Arc Expressions

The set of all arc expressions is defined as Δ, and an arc expression follows the following syntax:

$$\delta ::= n'(\tau) \mid n'(\sigma.all) \mid \delta \uplus \delta \mid \delta \setminus \delta \mid n * \delta$$

Where $n \in \mathbb{N}$, τ is a color expression, and $\sigma \in \Sigma$.

3.4.1 Semantics

To describe the semantics for arc expressions, we first need to define the function: $[\![]\!] : \Delta \times \mathbb{B} \to \mathbb{C}_{MS}$, which given an arc expression and a binding, returns a multi-set over colors in $\mathbb{C}$.

Definition 9. (Arc semantics)

$[\![(n'(\tau))\langle b\rangle]\!] = n'([\![(\tau)\langle b\rangle]\!])$	(number-of)
$[\![(n'(\rho.all))\langle b\rangle]\!] = \sum_{c\in\rho}[\![(n'(c))\langle b\rangle]\!]$	(all)
$[\![(\delta_1 \uplus \delta_2)\langle b\rangle]\!] = [\![(\delta_1)\langle b\rangle]\!] \uplus [\![(\delta_2)\langle b\rangle]\!]$	(sum)
$[\![(\delta_1 \setminus \delta_2)\langle b\rangle]\!] = [\![(\delta_1)\langle b\rangle]\!] \setminus [\![(\delta_2)\langle b\rangle]\!]$	(subtraction)
$[\![(n * \delta)\langle b\rangle]\!] = n * [\![(\delta)\langle b\rangle]\!]$	(scalar-product)

To give an example of an arc expression, we have the expression a defined as $1'(x)+2'(3)$. In this expression, the variables are defined as $Var(a) = \{x\}$. We then have $Type(x) = \{1, 2, 3\}$. This could then be evaluated under a binding such that $[\![a\langle x = 1\rangle]\!] = 1'(1) + 2'(3)$.

3.5 Definition of Colored Petri Nets with Inhibitor Arcs

In this subsection we define colored Petri nets with inhibitor arcs, using the concepts described until now.

Definition 10. (CPN)
A **Colored Petri Net with Inhibitor Arcs** is a nine tuple $\mathbf{CPN} = (\Sigma, P, T, C, G, F, W, I, W^I)$ where:

1. Σ is a finite set of finite **color sets**,
2. P is a finite set of **places**,
3. T is a finite set of **transitions**,
4. C is a **color function**, defined from P into Σ, i.e. $C : P \to \Sigma$,
5. G is a **guard** function, i.e. $G : T \to \Gamma$,
6. $F \subseteq (P \times T) \cup (T \times P)$ is the set of **arcs**,
7. W is an **arc expression** function, i.e. $W : F \to \Delta$,
8. $I \subseteq P \times T$ is the set of **inhibitor arcs**, and
9. W^I is an **inhibitor arc expression** function, i.e. $W^I : I \to \Delta$.

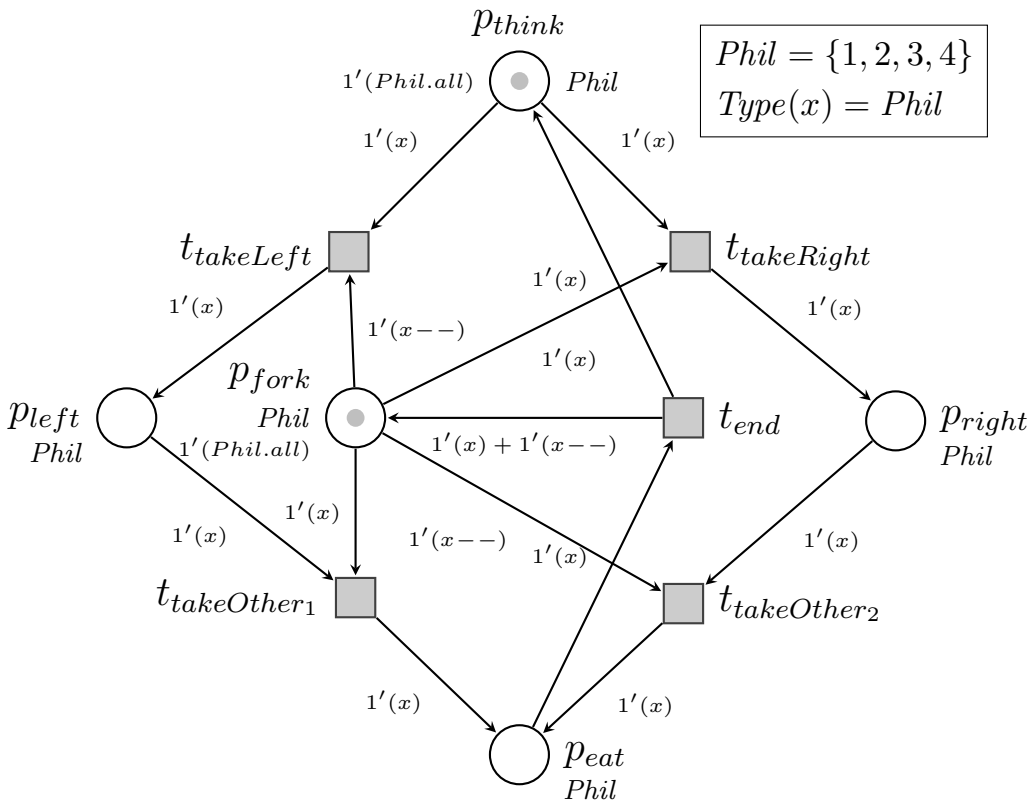
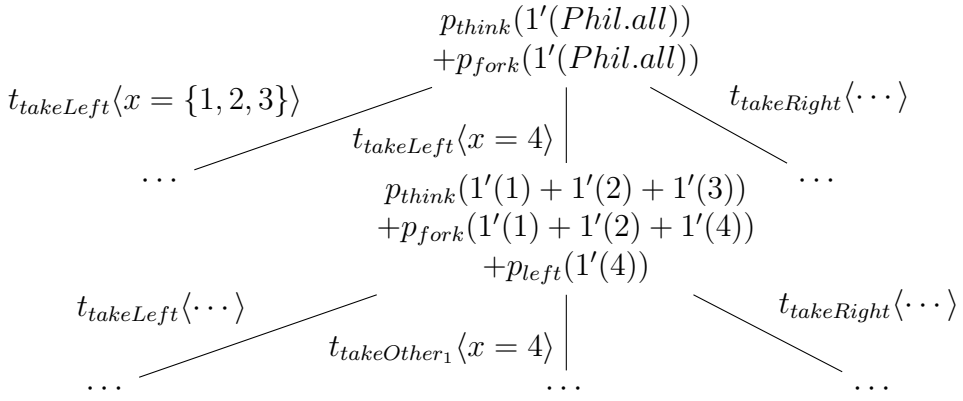


Figure 2: Dining philosophers modelled in a colored Petri net.



An example of a colored Petri net can be found in Figure 2. A fragment of the LTS for this, can be seen above. We go more in detail with the behavior

of this later. This net is a representation of the dining philosophers, which illustrates an inappropriate use of shared resources, generating deadlocks. This model is also a good example of a Petri net, which is easy to represent in a colored Petri net, but explodes in net size when modelled in uncolored Petri nets.

If this net has to be represented as an uncolored Petri net, we would have to represent each place as four individual places. We will see the general form of this in Section 3.8.

We now define the behavior of colored Petri nets with inhibitor arcs, before continuing with this example.

3.6 Dynamic behavior of Colored Petri Nets

First we need to define the function $Var(t)$ which returns the set of variables used in either the guard expression or in the arc expression of any of the connected arcs, i.e.

$$\begin{aligned}\forall t \in T : Var(t) = \{v \mid & v \in Var(G(t)) \\ & \vee \exists (p,t) \in F : v \in Var(W(p,t)) \\ & \vee \exists (t,p) \in F : v \in Var(W(t,p))\}\end{aligned}$$

Definition 11. (Bindings)
For a transition $t \in T$ with variables $Var(t) = \{v_1, v_2, \ldots, v_n\}$ we define the binding type $BT(t)$:

$$BT(t) = Type(v_1) \times Type(v_2) \times \ldots \times Type(v_n)$$

Also we define the set of all bindings $B(t)$:

$$B(t) = \{(c_1, c_2, \ldots, c_n) \in BT(t) \mid G(t)\langle v_1 = c_1, v_2 = c_2, \ldots, v_n = c_n\rangle\}$$

Which is the set of all bindings for a transition, that satisfy the guard of that transition, under the binding.

Next we need to define markings and steps of a colored Petri net:

Definition 12. (Marking)
A **marking** of a colored Petri net is a function $M : P \to C(p)_{MS}$ where $p \in P$.

We define the set of all markings as $\mathbb{M}$.

Before defining enabledness of a transition, we must first introduce the function $Inhib_\infty$. This function takes a multiset as input, and returns a multiset. This function is defined for all x in the input b:

$$Inhib_\infty(b)(x) = \begin{cases} \infty, & \text{if } b(x) = 0 \\ b(x) & \text{otherwise.} \end{cases}$$

The resulting multiset thus has infinite members of the types where it previously had none.

Definition 13. (Enabledness)
A transition $t \in T$ is **enabled** under binding $b \in B(t)$ in a marking M iff the following properties are satisfied:

1. $\forall (p,t) \in F : M(p) \geq W(p,t)\langle b \rangle$
2. $\forall (p,t) \in I : M(p) < Inhib_\infty(W^I(p,t)\langle b \rangle)$

Note that in Definition 13 that the marking multiset of place p has to be a subset of the multiset defined by the output of the function $Inhib_\infty$. This means that if the marking of place p contains at least the number of tokens of a given color than defined in the inhibitor arc weight function, then this inhibitor arc inhibits the transition. In [1] and [5] they use inclusion of the multiset instead of subset. Using the subset allows for better compatibility with unfolding to uncolored Petri nets, as we will see in Section 3.8.

Definition 14. (Transition firing)
When a transition $t \in T$ under binding $b \in B(t)$ is enabled in a marking M_1 it may fire, changing the marking M_1 to marking M_2, which is defined by:

$$\forall p \in P : M_2(p) = (M_1(p) \setminus W(p,t)\langle b \rangle) \uplus W(t,p)\langle b \rangle$$

M_2 is said to be directly reachable from M_1 by the firing of transition t, denoted by:

$$M_1 \xrightarrow{t} M_2.$$

Looking back at the net in Figure 2, we have a place for thinking philosophers in place p_{think}, and a place p_{fork} for storing forks. Both of these places have the color type *Phil*, where each color represents a dining philosopher. In place p_{fork} we do not define a new color type, since there exists one for each philosopher, and we thereby refer to the fork to the right of a philosopher x

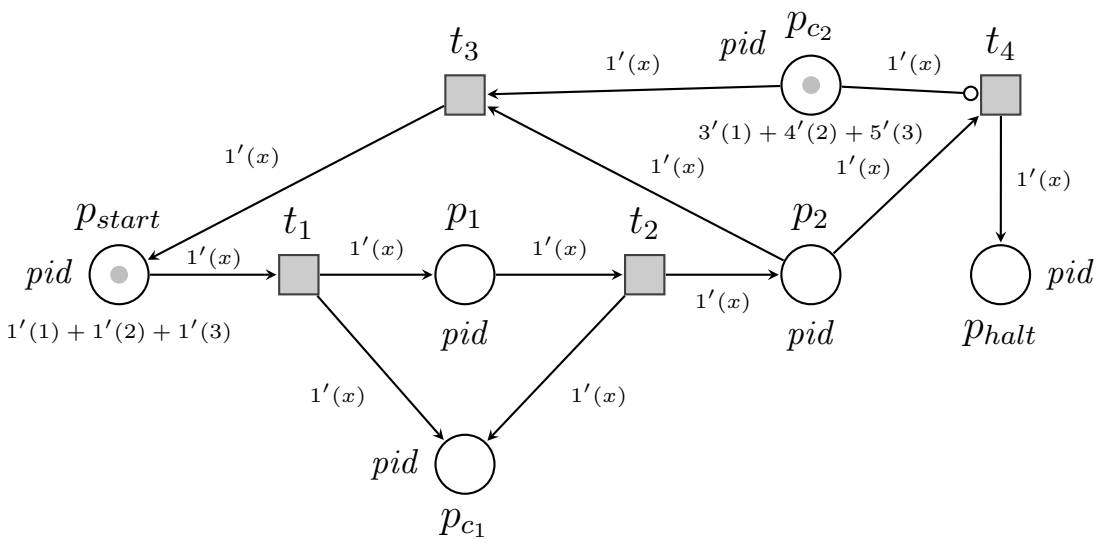
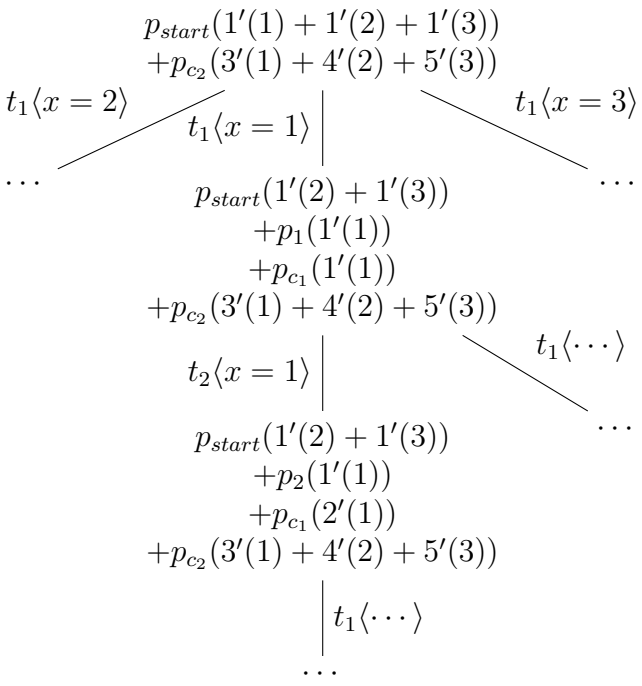


Figure 3: Colored Petri net simulating two-counter Minsky machine, calculating multiple of twos, in parallel

as fork x, and the fork on the left is $x--$. A thinking philosopher can now pick up either the fork to his right, or he can pick up the fork on his left, where they are located in p_{fork}. When a philosopher has one fork, then he can pick up the other fork, if the fork is available. When he is done eating, he can return both of the forks and return to thinking.



In Figure 3 we see another example of a colored Petri net where this includes inhibitor arcs. A fragment of the LTS can be seen above. This net represents a two-counter Minsky machine, running the following program:

Listing 1: Program running in Figure 3

```
1 c1 := c1 + 1; goto 2 /* p start */
2 c1 := c1 + 1; goto 3 /* p 1 */
3 if c2 > 0 then (c2 := c2 - 1; goto 1) else goto 4 /* p 2 */
4 HALT /* p halt */
```

The net in Figure 3 is running the program in three concurrent processes, where they each have different inputs, i.e. the process with *pid* 1 is running with the input 3, while process 2 is running with the input 4, and process 3 with input 5. Since we have added inhibitor arcs, we now have the ability to model any Turing complete computation, such as a net like this. We have that $\mathit{Type}(\mathit{pid}) = \{1, 2, 3\}$, which represents the process id of each process. Then we have the program counter represented by a token in the place corresponding to the code line (place names seen in the comments), and the two registers are represented by place p_{c_1} and p_{c_2}, and the number of tokens of a given process id corresponds to the register of that process. As such, each process will have its input in p_{c_2} and its output in p_{c_1}. An example path could be firing transition $t_1\langle x = 1\rangle$, transitioning the marking in p_{start} to $1'(2) + 1'(3)$ and adding a $1'(1)$ token to both p_1 and p_{c_1}, then

firing transition $t_2\langle x = 1\rangle$, adding another token to place p_{c_1}, and moving the program counter token to p_2. Now we can only fire transition $t_3\langle x = 1\rangle$, $t_1\langle x = 2\rangle$, or $t_1\langle x = 3\rangle$, since we still have tokens of color 1 in p_{c_1}, which inhibits $t_4\langle x = 1\rangle$. We can then fire transition $t_3\langle x = 1\rangle$, finishing the first loop of adding two tokens to the p_{c_1} counter. This loop can be repeated three times for *pid* 1, before ending in p_{halt}.

3.7 Querying Colored Petri Nets

In this thesis, we are using CTL [2] queries to query the colored Petri nets, but we do not go in-depth with CTL semantics. The CTL syntax used in this thesis is as follows:

$$\begin{aligned}\varphi ::= {} & \alpha \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid AX\varphi \mid EX\varphi \\ & \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A(\varphi_1 U \varphi_2) \mid E(\varphi_1 U \varphi_2)\end{aligned}$$

The set of all φ is denoted by CTL.

We will now have a closer look at the atomic propositions α used in the CTL queries on colored Petri nets. An atomic proposition over a colored Petri net, compares the number of tokens in a given place to either a constant integer number, or to another place. In the MCC [13] competition these comparisons are only on the total amount of tokens in a place, regardless of the colors of the tokens, and thus it is the type of comparisons we will focus on in this thesis. We are also able to ask whether a transition is fireable.

The syntax of these atomic propositions is:

$$\begin{aligned}\alpha &::= \mathit{true} \mid \mathit{false} \mid \beta \mid t \mid \mathit{deadlock} \\ \beta &::= v_1 < v_2 \mid v_1 \leq v_2 \mid v_1 > v_2 \mid v_1 \geq v_2 \mid v_1 = v_2 \mid v_1 \neq v_2 \\ v &::= p \mid n \mid v_1 + v_2 \mid v_1 - v_2 \mid v_1 \cdot v_2\end{aligned}$$

Where p is the name of a place, n is an integer constant, and t is transition fireability in a given marking. The set of all v expressions is denoted by V, and the set of all α is denoted by A.

In order to define the semantics, we must first establish some functions. The first is $[\![]\!]_M : V \times M \to \mathbb{N}^0$. The second function is $[\![]\!]_M : A \times M \to \{true, false\}$.

The semantics for the atomic propositions are defined as following:

$$
\begin{array}{ll}
[\![p]\!]_M = |M(p)| & \text{(place)} \\
[\![n]\!]_M = n & \text{(constant)} \\
[\![v_1 + v_2]\!]_M = [\![v_1]\!]_M + [\![v_2]\!]_M & \text{(addition)} \\
[\![v_1 - v_2]\!]_M = [\![v_1]\!]_M - [\![v_2]\!]_M & \text{(subtraction)} \\
[\![v_1 \cdot v_2]\!]_M = [\![v_1]\!]_M \cdot [\![v_2]\!]_M & \text{(multiplication)}
\end{array}
$$

$$
\begin{array}{ll}
[\![v_1 < v_2]\!]_M = [\![v_1]\!]_M < [\![v_2]\!]_M & \text{(less)} \\
[\![v_1 \leq v_2]\!]_M = [\![v_1]\!]_M \leq [\![v_2]\!]_M & \text{(less-eq)} \\
[\![v_1 > v_2]\!]_M = [\![v_1]\!]_M > [\![v_2]\!]_M & \text{(greater)} \\
[\![v_1 \geq v_2]\!]_M = [\![v_1]\!]_M \geq [\![v_2]\!]_M & \text{(greater-eq)} \\
[\![v_1 = v_2]\!]_M = [\![v_1]\!]_M = [\![v_2]\!]_M & \text{(eq)} \\
[\![v_1 \neq v_2]\!]_M = [\![v_1]\!]_M \neq [\![v_2]\!]_M & \text{(not-eq)}
\end{array}
$$

$$
[\![t]\!]_M = \begin{cases} true, & \text{if } t \text{ is enabled for some binding in marking } M \\ false, & \text{otherwise} \end{cases} \quad \text{(fireability)}
$$

3.8 Unfolding of Colored Petri Nets

In this section we describe the process of unfolding a colored Petri net with inhibitor arcs into an uncolored Petri net with inhibitor arcs. We start by first introducing the definition of the unfolding, followed by a theorem of bisimulation and a proof of this theorem.

Definition 15. (Unfolding)
Let $N = (\Sigma, P, T, C, G, F, W, I, W^I)$ be a colored Petri net. The unfolded net is a Petri net $N_u = (P_u, T_u, F_u, W_u, I_u, W_u^I)$, obtained by the unfolding of N such that:

1. $P_u = \bigcup_{p \in P} \bigcup_{c \in C(p)} (p, c)$,
2. $T_u = \bigcup_{t \in T} \bigcup_{b \in B(t)} (t, b)$,
3. $F_u = \{((p, c), (t, b)) \in P_u \times T_u \mid (W(p, t)\langle b \rangle)(c) > 0\} \cup \{((t, b), (p, c)) \in T_u \times P_u \mid (W(t, p)\langle b \rangle)(c) > 0\}$,
4. $\forall((p, c), (t, b)) \in F_u \cap (P_u \times T_u) : W_u((p, c), (t, b)) = (W(p, t)\langle b \rangle)(c)$ and $\forall((t, b), (p, c)) \in F_u \cap (T_u \times P_u) : W_u((t, b), (p, c)) = (W(t, p)\langle b \rangle)(c)$,
5. $I_u = \{((p, c), (t, b)) \in P_u \times T_u \mid (W^I(p, t)\langle b \rangle)(c) > 0\}$, and
6. $\forall((p, c), (t, b)) \in I_u : W_u^I((p, c), (t, b)) = (W^I(p, t)\langle b \rangle)(c)$.

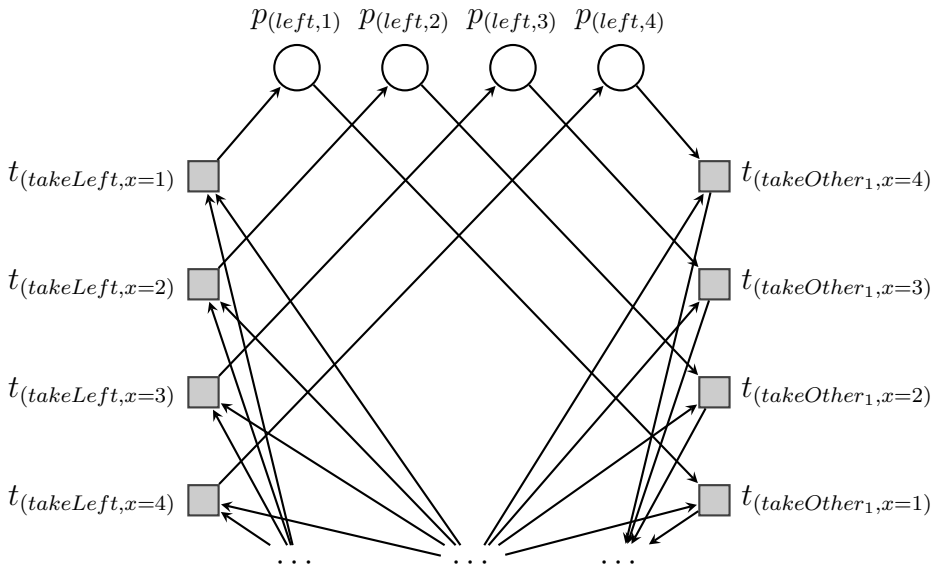


Figure 4: Unfolded fragment of Figure 2.

In Figure 4 we see a fragment of the net in Figure 2. Here we show the place p_{left} and the transitions $t_{takeLeft}$ and $t_{takeOther_1}$ unfolded. In this fragment, we see that representing only one place and two transitions, has more nodes than in the original net.

Next we show the equivalence between the colored Petri net and the unfolded Petri net, starting with the definition of marking equivalence.

Definition 16. (Marking equivalence)
Given marking M, we define the function $u : \mathbb{M} \to \mathbb{M}_u$ such that:

$$u(M)((p, c)) = M(p)(c)$$

Theorem 1.
Let N be a colored Petri net and N_u the unfolded Petri net of N. If $M \xrightarrow{t} M'$ under binding b, then $u(M) \overset{(t,b)}{\to} u(M')$, and if $u(M) \overset{(t,b)}{\to} u(M')$, then $M \xrightarrow{t} M'$ under binding b.

Proof 1.
Assume $M \xrightarrow{t} M'$ under binding b in a colored Petri net N. Then let N_u be the unfolded Petri net of N.

For each binding $b \in B(t)$ for transition t in N, there exists a corresponding transition (t, b) in N_u, as per definition 15. For each arc connected to

a transition t in N, there exists a corresponding set of arcs to the transition (t, b) in N_u with corresponding weights, as per Definition 15. We know that transition (t, b) is enabled iff there is enough tokens in the marking of each place to satisfy the in-going arcs, and that there is less tokens in the marking of each place than the threshold defined for the connecting inhibitor arc. Then firing transition (t, b) in N_u will change marking $u(M)$ to $u(M')$, where:

$$u(M')((p, c)) = u(M)((p, c)) - W_u((p, c), (t, b)) + W_u((t, b), (p, c)) \quad (1)$$

We know as per Definition 15, that transition t is only enabled, if each place connected by an inhibitor arc, has less tokens of the colors specified in the arc expression. Since $u(M)((p, c)) = M(p)(c)$, and for all in-going arcs to transition (t, b), $W_u((p, c), (t, b)) = (W(p, t)\langle b\rangle)(c)$ as per Definition 15, and for all out-going arcs from transition (t, b), $W_u((t, b), (p, c)) = (W(t, p)\langle b\rangle)(c)$ also as per Definition 15. Since:

$$M'(p)(c) = M(p)(c) - (W(p, t)\langle b\rangle)(c) + (W(t, p)\langle b\rangle)(c) \quad (2)$$

Then $M'(p)(c) = u(M')(p, c)$.

Assume $u(M) \overset{(t,b)}{\rightarrow} u(M')$ in an unfolded Petri net N_u, unfolded from a colored Petri net N. Then from Definition 15 we know there exists a transition t which can be fired under binding b. Given Equation (1) and Equation (2), we find the relation $u(M')(p, c) = M'(p)(c)$, following the same logic as before. □

3.9 Translating Colored Petri Net Queries

Since we have shown that a colored Petri net N can be unfolded into a bisimilar Petri net N_u, and we know that CTL queries can not distinguish between bisimilar behavior [3], we just have to translate the atomic propositions to the equivalent states in the unfolded Petri net N_u.

Since we only use total token count in places in our atomic propositions, the same syntax and semantics can be used to express atomic propositions over uncolored Petri nets.

In order to translate query φ, we have to define the function $\mathit{unfold}(\varphi) : CTL \rightarrow CTL$. The function is defined as:

$unfold(p) = \sum_{c \in C(p)}(p, c)$
$unfold(t) = \bigvee_{b \in B(t)}(t, b)$
$unfold(\varphi_1\ op\ \varphi_2) = unfold(\varphi_1)\ op\ unfold(\varphi_2)$
$unfold(\neg\varphi) = \neg unfold(\varphi)$
$unfold(AX\varphi) = AX unfold(\varphi)$
$unfold(AF\varphi) = AF unfold(\varphi)$
$unfold(AG\varphi) = AG unfold(\varphi)$
$unfold(EX\varphi) = EX unfold(\varphi)$
$unfold(EF\varphi) = EF unfold(\varphi)$
$unfold(EG\varphi) = EG unfold(\varphi)$
$unfold(A(\varphi_1 U \varphi_2)) = A(unfold(\varphi_1) U\, unfold(\varphi_2))$
$unfold(E(\varphi_1 U \varphi_2)) = E(unfold(\varphi_1) U\, unfold(\varphi_2))$

Given this function we can now formulate the following theorem:

Theorem 2.
Given a colored Petri net N, a query φ, and an unfolded Petri net N_u then $N \vDash \varphi$ iff $N_u \vDash unfold(\varphi)$.

Proof 2.
To prove Theorem 2 we know that the only difference between φ and $unfold(\varphi)$ is the case of p and t. Starting with p, the definition of p under a binding M in a colored Petri net is $|M(p)|$. Since the definition of the cardinality of that multiset is equal to the sum of all occurrences, we know that sum of all occurrences in the unfolded net is $\sum_{c \in C(p)}(p, c)$.

Next looking at t, we find that the fireability of t is true if it can fire under some binding. Then in the unfolded net, we have bound each binding to its own transition. Thus we just need to be able to fire at least one of these bindings. This can be expressed as $\bigvee_{b \in B(t)}(t, b)$.

Lastly we know that CTL cannot distinguish between bisimilar states [3], and that we have only changed the atomic propositions in order for them to be equivalent in the unfolded net, then by Theorem 1 we can conclude that the CTL query φ is equivalent in the unfolded net. □

4 CPN Overapproximation

In this section we show a method for overapproximating an answer for a class of queries that can be queried on a colored Petri net. First we define a method for calculating the cardinality of an arc expression without evaluating it, which is used in a method for stripping a colored Petri net of its colors. This method leaves us with a stripped uncolored Petri net. The method is

defined thereafter. Next we define the set of queries we can answer using a stripped Petri net. Then we define an algorithm for running the overapproximation, which also shows which situations we can use the results and when we cannot. Later, we present results from testing this method, and compare the execution time of the solved queries to an engine which unfolds the net, and then solves the queries for the unfolded net. Both these methods are implemented in the tool *verifypn* [2] from the tool collection *TAPAAL* [6].

4.1 Arc Expression Cardinality

Before moving to the stripping of colored Petri nets, we must first define a way to find the cardinality of an arc expression without a binding. To do this, we define the partial function $size : \Delta \hookrightarrow \mathbb{N}$.

Given the arc expression syntax:

$$\delta ::= n'(\tau) \mid n'(\sigma.all) \mid \delta_1 \uplus \delta_2 \mid \delta_1 \setminus \delta_2 \mid n * \delta$$

We now define the function as following:

$size(n'(\tau)) = n$	(size-number-of)
$size(n'(\sigma.all)) = n * \lvert\sigma\rvert$	(size-all)
$size(\delta_1 \uplus \delta_2) = size(\delta_1) + size(\delta_2)$	(size-sum)
$size(n * \delta) = n * size(\delta)$	(size-scalar)

The *size* function for $\delta_1 \setminus \delta_2$ is a little more complicated, as this actually requires some knowledge of the binding, in order to correctly calculate the size, since subtraction on multisets only subtracts elements that are already in the multiset on the left hand side, thus not always being a direct subtraction of the two constituents.

We can only define the function for certain sub-expressions. This leads to the following definition:

$$size(\delta_1 \setminus \delta_2) = \begin{cases} n * \lvert\sigma\rvert - min(m, n), & \text{if } \delta_1 \equiv n'(\sigma.all) \wedge \delta_2 \equiv m'(\tau) \\ undefined & \text{otherwise.} \end{cases}$$

If any sub-expression of the *size* function returns *undefined*, then they return *undefined*.

Theorem 3.
For all $\delta \in \Delta$, if $size(\delta)$ is defined, then $size(\delta) = \lvert\delta\langle b\rangle\rvert$ for all bindings b.

Proof 3.
To prove Theorem 3, we use structural induction on δ. Here we have to

prove that Theorem 3 holds in 1) $\delta = n'(\tau)$, 2) $\delta = n'(\sigma.all)$, 3) $\delta = \delta_1 \uplus \delta_2$, 4) $\delta = n * \delta$, and 5) $\delta = \delta_1 \setminus \delta_2$. We now let b be a binding.

Starting with 1), the cardinality of the multiset from the expression of $\delta\langle b \rangle$ is always n. Thus $size(n'(\tau)) = n = |n'(\tau)\langle b \rangle|$. Looking at 2), we have that $|n'(\sigma.all)\langle b \rangle| = n * |\sigma| = size(n'(\sigma.all))$.

Moving on to 3), then we have that if $size(\delta_1) = |\delta_1\langle b \rangle|$ and $size(\delta_2) = |\delta_2\langle b \rangle|$, then $size(\delta_1 \uplus \delta_2) = size(\delta_1) + size(\delta_2)$.

Next, we look at 4), where we see that if $size(\delta) = |\delta\langle b \rangle|$, then $size(n*\delta) = n * size(\delta) = |n * \delta\langle b \rangle|$.

Lastly, we have 5). Here it is only defined, if $\delta_1 = n'(\sigma.all)$ and $\delta_2 = m'(\tau)$. As such, $size(n'(\sigma.all) \backslash m'(\tau)) = n*|\sigma\langle b \rangle| - min(m, n) = |n'(\sigma.all) \backslash m'(\tau)\langle b \rangle|$. This is because when doing a multiset subtraction, we can only subtract the amount of elements, that is already in the multiset, hence we can at most subtract n elements, of each type from the multiset. □

4.2 Stripping Colored Petri Nets

For our overapproximation algorithm, we strip the net of its colors. The stripped net has the same places, transitions, and arcs, as the original colored Petri net. All the transitions have their guards removed, which allows for more behavior than the colored Petri net, hence this being an overapproximation. In addition to stripping the guards, we also strip the colors from arc expressions, by using the cardinality of the multiset generated by them. We can do this because no matter the binding of an arc expression, the cardinality is always the same.

A stripped net can then be defined as:

Definition 17. (Stripping)
Given a colored Petri net $N = (\Sigma, P, T, C, G, F, W, I, W^I)$ such that for all arc expressions *size* is defined, then a stripped Petri net $N^S = (P^S, T^S, F^S, W^S, I^S, W^{IS})$ is defined by:

1. $P^S = P$
2. $T^S = T$
3. $F^S = F$
4. $\forall (p,t) \in F \cap P \times T : W^S(p,t) = size(W(p,t))$ and $\forall (t,p) \in F \cap T \times P : W^S(t,p) = size(W(t,p))$
5. $I^S = I$

6. $\forall(p,t) \in I : W^{IS}(p,t) = size(W^I(p,t))$

We can then define the following definition for marking equivalence:

Definition 18.
Given a CPN N with marking M, and a stripped Petri net N^S with marking M^S, we define $M \equiv M^S$ iff for all $p \in P$ holds $M^S(p) = |M(p)|$.

The advantages of stripping versus unfolding is the vastly reduced statespace compared to most unfolded Petri nets. This results in faster verification times. The problem however is the number of queries that returns inconclusive answers using stripped Petri nets.

4.3 Approximation Preserving Logic

Since the stripped Petri net can only answer a subset of queries, we need to define the subset of queries that can be answered by the stripped Petri net. In this subsection we assume that all φ queries have been normalized by the function $pushNeg(\varphi)$, which normalizes the logic with the use of De Morgan's law, pushing the negations to the atomic predicates.

First we define the Approximation Preserving Logic (APL):

$$\psi ::= true \mid false \mid t \mid \neg deadlock \mid \beta \mid \neg\beta \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2$$
$$\beta ::= v_1 < v_2 \mid v_1 \leq v_2 \mid v_1 > v_2 \mid v_1 \geq v_2 \mid v_1 = v_2 \mid v_1 \neq v_2$$
$$v ::= p \mid n \mid v_1 + v_2 \mid v_1 - v_2 \mid v_1 \cdot v_2$$

With this logic we seek to capture as many aspects from the colored Petri net as possible, while preserving correctness. Correctness in this case is defined as being able to simulate the same behavior in both nets. As such we see that β, t, and $deadlock$ are as defined in Subsection 3.7, but note that we do not allow $\neg t$ and $deadlock$. These queries cannot be answered faithfully by the stripped net. An example of a net that would not give a correct answer to a query like $EF\neg t$ can be seen in Figure 5. Here we see that in this case the query would evaluate to true in the colored Petri net, but not in the stripped net. This example also shows us that if there is a deadlock in the colored Petri net, it is not necessarily present in the stripped net.

We can now define set of all expressions as APL. We then see that $APL \subset CTL$.

(a) Colored Petri net

(b) Stripped Petri net

Figure 5: Counter example of $EF\neg t$.

Given this logic we can form the following lemma:

Lemma 1.
Let $N = (\Sigma, P, T, C, G, F, W, I, W^I)$ where $I = \emptyset$ with marking M, N^S a stripped net of N with marking M^S, let $M \equiv M^S$, and let $\psi \in APL$. If $M \vDash \psi$ then $M^S \vDash \psi$.

Proof 4.
We can show this implication by structural induction on ψ. Thus we need to show that Lemma 1 holds in 1) $\psi = \mathit{true}$, 2) $\psi = \mathit{false}$, 3) $\psi = \beta$, 4) $\psi = \neg\beta$, 5) $\psi = t$, 6) $\psi = \neg \mathit{deadlock}$, 7) $\psi = \psi_1 \wedge \psi_2$, and 8) $\psi = \psi_1 \vee \psi_2$.

When looking at 1) and 2), we see that these cases are trivial. Moving on to 3), we have several expressions in β, but we only need to look at one of the subexpressions, i.e. p, which then renders the rest of the subexpressions, i.e. p, which then renders the rest of the subexpressions trivial. In order to show that $[\![p]\!]_M = [\![p]\!]_{M^S}$, we look at the how $[\![p]\!]_M$ and $[\![p]\!]_{M^S}$ is defined. We see that $[\![p]\!]_M = |M(p)|$ and $[\![p]\!]_{M^S} = M^S(p)$, then looking at Definition 1 we find that $M^S(p) = |M(p)|$, hence the lemma holds for 3). The same holds for $\neg\beta$.

In 5), we assume that $M \vDash t$, then from Definition 13 we know that in order for this transition to be enabled, then we satisfy the first property $\forall(p,t) \in F : M(p) \geq W(p,t)\langle b \rangle$. Since we do not have any inhibitor arcs we always satisfy the second property. For uncolored Petri nets, we know from Definition 18, that since $M^S \equiv M$, then $M^S(p) = |M(p)|$. From Definition 3 we know that t is enabled if $\forall(p,t) \in F : M^S(p) \geq W^S((p,t))$. We also know from Definition 17 that each arc has the same weight in the stripped net, as the cardinality of the arc expression in the colored net. Thus we know that if $M \vDash t$, then $M^S \vDash t$.

For the query type 6), we know that the definition for a deadlock is that there is no enabled transitions. We then assume that $M \vDash \neg \mathit{deadlock}$, then we know that there must exist a transition that is enabled in the colored

Petri net under marking M. From 5), we know that if a transition is enabled under M, then the corresponding transition is enabled under M^S. Therefore we know that if $M \vDash \neg deadlock$, then $M^S \vDash \neg deadlock$.

For 7) we have that $M \vDash \psi_1 \wedge \psi_2$, which implies $M \vDash \psi_1$ and $M \vDash \psi_2$. Then following the structural induction, we know that $M^S \vDash \psi_1$ and that $M^S \vDash \psi_2$. This implies that $M^S \vDash \psi_1 \wedge \psi_2$.

Lastly 8) we see that $M \vDash \psi_1 \vee \psi_2$, which implies that $M \vDash \psi_1$ or $M \vDash \psi_2$. Then following the structural induction, we know that either $M^S \vDash \psi_1$ or $M^S \vDash \psi_2$. From this, we imply $M^S \vDash \psi_1 \vee \psi_2$. □

Now given Lemma 1 and Definition 18, we come to the last lemma:

Lemma 2.
Let $N = (\Sigma, P, T, C, G, F, W, I, W^I)$ where $I = \emptyset$ with marking M_0, N^S a stripped net of N with marking M_0^S, then let $M_0 \equiv M_0^S$. If $M_0 \rightarrow^* M$ and $M \vDash \psi$, then $M_0^S \rightarrow^* M^S$ and $M^S \vDash \psi$.

This holds because any given marking we can transition to, from the initial marking in a colored Petri net, we can transition to in an equivalent marking in the stripped Petri net, but not the other way around.

Theorem 4.
Let $N = (\Sigma, P, T, C, G, F, W, I, W^I)$ where $I = \emptyset$, with marking M_0, and let N^S be a stripped Petri net with an equivalent marking M_0^S. If $M_0 \vDash EF\psi$, then $M_0^S \vDash EF\psi$.

Now we can express $EF\psi$, but we still lack $AG\psi$. This can also be expressed as $\neg EF\neg\psi$. If we normalize $\neg\psi$, into ψ', and $\psi' \in APL$, then following Theorem 4 we know that $EF\psi'$ is a query we can answer using the stripped net. Since we negate EF, we must also negate which answers are correct, and which are undefined. By this reduction, we can also answer AG queries.

4.4 Interpreting The Results

Now we describe the pseudo code for how the overapproximation algorithm works. This algorithm takes a colored Petri net, a marking, and a query as input. The algorithm then returns whether the marking M satisfies the

query φ or not, or returns inconclusive.

Data: CPN $N = (\Sigma, P, T, C, G, F, W, I, W^I)$, Marking M, Query φ
Result: $M \vDash \varphi$, $M \nvDash \varphi$, or *Inconclusive result*
if φ *is not* $EF\varphi'$ *or* $AG\varphi'$ **then**
  return *Inconclusive result*;
end
if N *contains arc expression with undefined size, or* $I \neq \emptyset$ **then**
  return *Inconclusive result*;
end
$N^S :=$ strip N;
$M^S :=$ strip marking M;
if $\varphi \equiv EF\varphi'$ **then**
  $\psi := pushNeg(\varphi')$;
  if $\psi \notin APL$ *or* $M^S \vDash EF\psi$ **then**
    return *Inconclusive result*;
  end
  return *false*;
end
if $\varphi \equiv AG\varphi'$ **then**
  $\psi := pushNeg(\neg\varphi')$;
  if $\psi \notin APL$ *or* $M^S \vDash EF\psi$ **then**
    return *Inconclusive result*;
  end
  return *true*;
end

Algorithm 1: Interpreting query results

As we see from Algorithm 1, the class of queries we are able to answer conclusively is limited to Reachability queries, i.e. a reduced set of CTL queries, and only queries that do not contain deadlock nor negated fireability queries after being normalized.

5 Implementation

In this section, we discuss the implementation done in the tool *verifypn* [2]. The source code can be found at `https://launchpad.net/verifypn`, where the code is part of the release version. In this tool, we extended the parser to accommodate the additional extensions for colored Petri nets in the PNML standard [16]. As the standard currently has no definition of inhibitor arcs for colored Petri nets, this extension of colored Petri nets has not been implemented in the tool. Additionally we implemented the functionality to unfold

the new colored Petri net structure into the existing representation of Petri nets in the engine, along with the ability to run the overapproximation algorithm. For the remainder of this section, we lay out some of the design decisions.

5.1 Data Structures

As the design of the parsing of the PNML standard is mostly dependent on the data structures used for representing the colored Petri net, we only discuss these.

When having to represent colors, we are met with the challenge of product colors, which can be used as colors. To overcome this, we used the composite pattern. The real challenge was how to compare two product colors, as we had to choose between spacial complexity and temporal complexity, in order to either look up product colors in a sorted map, or compute the relations. In the end we chose to compute the relations, as the spacial consumption of some product color maps exploded.

As we see in Definition 11, we needed to compute the set of all bindings for each transition t, as this is needed for computing the unfolding of both transitions, and arcs. This structure is currently implemented as a list of all bindings that satisfy the guard expression of the transition t. The disadvantage of this method, is that each binding also contain invariant variables, which do not have any impact on the result of the evaluation of the guard expression. As such, we still store each combination of bindings, over all variables. This reduces the temporal complexity, but as mentioned above, this can also explode in spacial complexity. Thus it could be a possibility to test other data structures, which simplify spacial complexity at the cost of temporal complexity.

6 Experiments

In this section, we describe the two experiments we did, and show the testing results of the implementations. First we compare the unfolding speed of the implementation build into *verifypn*, and the unfolding speed of the implementation in the *MCC* tool. After this, we compare the number of queries we are able to answer using the overapproximation algorithm, compared to unfolding a net, and then verifying the unfolded net. We also test how running the overapproximation algorithm and unfolding with verification, consecutively compares to only running verification on an unfolded net.

6.1 Experimental Setup

All tests in this thesis were conducted on an AMD Opeteron 6376 processor running single-threaded with a memory limit of 15 GB, and a timeout after 20 minutes for each run. All experiments were run on the 136 nets in the MCC'2017 competition [13]. The version used in all the experiments can be found at `https://code.launchpad.net/~verifypn-cpn/verifypn/andreas-exam`.

6.2 Unfolding

In this subsection we to look at the results of the unfolding experiments. The full results of this experiment can be found in Appendix A and Appendix B. In these tables OOM is short for *Out Of Memory*, and TO is short for *Timed Out*.

6.2.1 Setup

The unfolding experiment consisted of two parts. The first part was running the modified version of *verifypn*, which can be found at the link mentioned in Subsection 6.1, which allowed us to skip the verification engine and only print the unfolded net. Here we collected the time it took for the unfolding, from the time the engine had parsed the input file, to the time we had an in-memory representation of the unfolded net. We also timed the whole execution. The second part was comprised of running a version of *MCC* containing a timer that timed the execution of the unfolding algorithm, which was also timed from the point it had parsed the input, to it having an internal representation of the unfolded net. This time was collected, along with the total execution time. We ran this tool with the option to output for the LoLA format [17]. Both of these parts were set to output to */dev/null* in order to avoid introducing drive bottlenecks.

6.2.2 Results

If we look at the average unfolding time of all the nets in the MCC'2017 competition, we find that our implementation takes on average 3.680 seconds, while the *MCC* tool takes on average 2.580 seconds. We see that in most nets, the *MCC* tool is faster than our implementation in the unfolding part of the run time. Especially in the *Philosophers-COL* nets we fall behind, as the *MCC* implementation has a focus on detecting nets with a single variable used with a circular symmetry.

Though the *MCC* tool is faster at unfolding it lacks behind in total run time, as this tool transforms the unfolded net into a string in either a *hlnet* format or a *LoLA* format. This procedure is very costly, and in six cases results in run times 300 seconds slower than in our implementation. In Appendix B, we find a table that shows the total time it took to finish each program running on each net. If we look at this table, we see that our implementation has a shorter run time in every net except *CSRepetitions-COL-02*, where the difference is less than 40 milliseconds. Since the *verifypn* tool supports to output the resulting Petri net, we ran it with this option, but since this tool does not natively support writing the output net without verification, we modified it to do this. Looking at a net like *SharedMemory-COL-000200*, we see that our unfolding time is 2.078 seconds, while *MCC* does it in 0.64 seconds, which is almost 1.5 seconds faster, but we finish writing the result in 11.63 seconds, whereas *MCC* finishes writing the output in 1105.57 seconds. Looking at *Philosophers-COL-010000*, which *MCC* is optimized for, we find that our implementation unfolds in 11.74 seconds, with a total run time of 16.08 seconds, where *MCC* unfolds in 2.32 seconds, but has a total run time of 171.68 seconds. There were also 11 executions were both tools unfolded the nets, but only our implementation finished within the time limit.

6.3 Overapproximation

In this subsection we describe at the setup of the overapproximation experiments, and look at the results.

6.3.1 Setup

The overapproximation experiment involved running three setups. For each setup we ran 32 queries for each net, which were from the same competition as the nets, where the queries were split evenly between cardinality and fireability queries, i.e. the categories *ReachabilityCardinality* and *ReachabilityFireability* in the competition. We did not include queries from the competition categories *ReachabilityDeadlock*, *CTLCardinality*, and *CTLFireability* as these would not be able to be solved by the overapproximation algorithm, unless some of the CTL queries happened to be belong the set *APL*. In the first setup, denoted *Unfolded*, we ran our implementation of unfolding, and then using the existing verification engine, we ran the verification on the unfolded net. This was done for each query for each net. We then collected the amount of queries that we were able to verify. The second setup, denoted *Overapproximation*, consisted of running the overapproximation al-

gorithm on every query for each net. We collected the number of queries we were able to verify using this algorithm. We also noted how many nets were exclusively solved by this setup in comparison to the *Unfolded* setup and how many were exclusively solved by the *Unfolded* setup in comparison to this setup. The third and last setup, denoted *Combined*, ran a script which first tried to run the overapproximation algorithm, and in case this returned inconclusive, then it ran the unfolding algorithm followed by verification of the unfolded net. This setup was also tested on each query for each net. We then collected the number of queries solved by this method.

6.3.2 Results

As seen from the algorithm in Section 4.4, there are some nets and some queries that we are not able to answer using the overapproximation algorithm.

ReachabilityCardinality		<0.1s	<1s	<5s	<30s	<60s	<20m
Unique	*Unfolded*	165	372	553	762	813	963
	Overapproximation	628	406	282	206	199	193
Total	*Unfolded*	436	882	1196	1490	1548	1704
	Overapproximation	899	916	925	934	934	934
	Combined	1046	1295	1486	1700	1749	1895

Table 1: Runtime brackets for *Unfolded*, *Overapproximation*, and *Combined* setups, with uniquely solved queries compared between *Unfolded* and *Overapproximation*, and total solved queries for all setups, using ReachabilityCardinality queries.

In Table 1 we see that the overapproximation algorithm finishes all the nets that it can answer within at most 30 seconds, and that within the first 0.1 second it has uniquely solved 628 queries, that is not solved by the unfolded verification engine in that time brackets. Though we see that the longer we wait the amount of answers found solely by the overapproximation is decreasing. When looking at the combined method, we see that this scales very closely with the amount of unique answers from the overapproximation algorithm, in comparison to the total amount of nets verified with only the unfolded verification.

Looking at Table 2, we see that the tendencies of Table 1 are also visible here. Though if we look at the less than five seconds bracket, then we see that we answered more queries with the combined script, than we did if we add the total unfolded field with the unique overapproximation field, and this is due to the runtime variance between runs.

ReachabilityFireability		<0.1s	<1s	<5s	<30s	<60s	<20m
Unique	*Unfolded*	187	307	438	690	1209	1433
	Overapproximation	110	103	93	85	72	68
Total	*Unfolded*	206	334	475	735	1267	1495
	Overapproximation	129	130	130	130	130	130
	Combined	304	437	578	828	1327	1550

Table 2: Runtime brackets for *Unfolded*, *Overapproximation*, and *Combined* setups, with uniquely solved queries compared between *Unfolded* and *Overapproximation*, and total solved queries for all setups, using ReachabilityFireability queries.

We also see that the number of queries that can be solved by the overapproximation algorithm is 804 queries less than for *ReachabilityCardinality* queries. When analyzing the outputs of the algorithm, we find that 1478 of the queries in the *ReachabilityFireability* category were not in *APL*, which might be a side effect of not allowing negated fireability queries. We also found that 7 of the nets contained arcs with undefined *size*.

From both Table 1 and 2 we see that the unfolding algorithm offers a speedup, in comparison to the verification of the unfolded net. This is because this algorithm generally is faster than unfolding a colored Petri net and verifying the larger unfolded net. Also this algorithm does not offer much overhead if it is not able to answer the query.

6.4 Summary

So in summary, our unfolding implementation is not as fast as an existing solution in the specifics of unfolding, but in overall usage outperforms *MCC*. Our implementation could be improved in a number of ways, in order to catch up to *MCC*, which we will touch more on later.

As for the overapproximation we see that this offers a significant speedup in those queries where applicable, as discussed in Subsection 6.3. The fact that it does not take much overhead, if it is not able to answer a query makes it a good strategy to try before running the unfolding, and then verifying the unfolded query on the unfolded net.

7 Conclusion

In this thesis, we defined colored Petri nets with inhibitor arcs and introduced a method for unfolding these into uncolored Petri nets with inhibitor arcs.

In addition to this we defined a method of unfolding CTL queries in order for them to fit the unfolded Petri nets. We also proposed an algorithm for overapproximating an answer to a subset of queries, without unfolding a net.

We then implemented both the unfolding algorithm, and the overapproximation algorithm in the tool *verifypn*. Following this we compared the speed of our unfolding implementation to an existing implementation in the tool *MCC*. In the unfolding alone we were on average 29.89% slower than this tool. This can possibly be counteracted by changing the generation of possible bindings for a transition, such that we do not have to compute invariant bindings for any transition or connected arc. Another approach to improve the performance is to implement an expression analyzer, which would be able to calculate the exact set of bindings needed for a transition without iterating through each combination. Here one could look into the *Z3* theorem prover[2] [7].

We also tested the overapproximation algorithm against unfolding a colored net and verifying the unfolded net. We found this to be an effective algorithm to use in conjunction with unfolding and then verifying the unfolded net. This offered both a speed boost in most cases, and even allowed for answering queries in nets that were otherwise very difficult to unfold.

In the future it would be interesting to look at structural net reductions on the colored nets, as this would also impact the unfolding, as fewer places and transition also allows for faster unfolding.

Acknowledgements. Lastly we would like to thank Peter Gjøl Jensen for his contribution with a counter example to why transition firing cannot be negated in the Approximation Preserving Logic, and for doing code reviews on the implementation.

A special thanks also goes out to Jiri Srba for excellent sparring during the making of this thesis.

8 Bibliographical Remarks

This thesis builds upon [12]. In [12] we made a prototype of the unfolding algorithm in Python, where we did not include unfolding of queries. For this thesis we rewrote the whole implementation in C++ and extended this to include queries as well. Subsection 3.1 is taken from [12], but extended with more explanations and examples. Subsection 2.2 is also taken from [12].

[2] `https://github.com/Z3Prover/z3`

In this thesis the definition of Petri nets has been based on [14], but extended with inhibitor arcs. The multiset theory is based on the one found in [9] and [1]. The base idea of different color types are based on [4], but the color data types are based on the PNML standard in [16]. The definition of colored Petri nets is based on the work in [9, 11], but extended with the inhibitor arcs. The definition of enabledness was originally inspired by [11] for colored Petri nets without inhibitor arcs, while the inspiration for the enabledness definition for inhibitor arcs came from [1]. The definition of the unfolding is based on [11], but has a novel addition for unfolding inhibitor arcs.

References

[1] Jonathan Billington. *Extending coloured petri nets*. Tech. rep. UCAM-CL-TR-148. University of Cambridge, Computer Laboratory, Sept. 1988. URL: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-148.pdf (cit. on pp. 2, 6, 14, 34).

[2] F.M. Boenneland et al. "Simplification of CTL Formulae for Efficient Model Checking of Petri Nets". In: *Proceedings of the 39th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets'18)*. LNCS. To appear. Springer-Verlag, 2018, pp. 1–20 (cit. on pp. 2, 17, 22, 27).

[3] Julian Bradfield and Colin Stirling. "Modal Mu-Calculi". In: (2005), pp. 13–14. URL: http://homepages.inf.ed.ac.uk/jcb/Research/MLH-bradstir.pdf (cit. on pp. 20, 21).

[4] G. Chiola et al. "On Well-Formed Coloured Nets and Their Symbolic Reachability Graph". In: *High-level Petri Nets: Theory and Application*. Ed. by Kurt Jensen and Grzegorz Rozenberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 373–396. ISBN: 978-3-642-84524-6. DOI: 10.1007/978-3-642-84524-6_13. URL: https://doi.org/10.1007/978-3-642-84524-6_13 (cit. on pp. 7, 34).

[5] Søren Christensen and Niels Damgaard Hansen. "Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs". In: *Application and Theory of Petri Nets 1993*. Ed. by Marco Ajmone Marsan. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 186–205. ISBN: 978-3-540-47759-4 (cit. on p. 14).

[6] Alexandre David et al. "TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets." In: *TACAS*. Vol. 12. Springer. 2012, pp. 492–497 (cit. on p. 22).

[7] Andreas Fröhlich et al. "Stochastic Local Search for Satisfiability Modulo Theories". In: AAAI, 2015. URL: https://www.microsoft.com/en-us/research/publication/stochastic-local-search-for-satisfiability-modulo-theories/ (cit. on p. 33).

[8] M. Hack. *PETRI NET LANGUAGE*. Tech. rep. Cambridge, MA, USA, 1976 (cit. on pp. 1, 3).

[9] Kurt Jensen. "Coloured petri nets: A high level language for system design and analysis". In: *Advances in Petri Nets 1990*. Ed. by Grzegorz Rozenberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 342–416. ISBN: 978-3-540-46369-6. DOI: 10.1007/3-540-53863-1_31. URL: https://doi.org/10.1007/3-540-53863-1_31 (cit. on pp. 2, 6, 34).

[10] Kurt Jensen. "Coloured petri nets and the invariant-method". In: *Theoretical Computer Science* 14.3 (1981), pp. 317 –336. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(81)90049-9. URL: http://www.sciencedirect.com/science/article/pii/0304397581900499 (cit. on pp. 1, 2).

[11] Kurt Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*. Vol. 1. Springer Science & Business Media, 2013 (cit. on pp. 2, 34).

[12] Andreas H. Klostergaard. "Unfolding of High Level Symmetric Nets". In: (2018) (cit. on p. 33).

[13] F. Kordon et al. *Complete Results for the 2017 Edition of the Model Checking Contest*. http://mcc.lip6.fr/2017/results.php. 2017. (Visited on 2017) (cit. on pp. 2, 7, 17, 29).

[14] Carl Adam Petri. "Kommunikation mit automaten". In: (1962) (cit. on pp. 1, 3, 34).

[15] C. V. Ramamoorthy and G. S. Ho. "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets". In: *IEEE Transactions on Software Engineering* SE-6.5 (1980), pp. 440–449. ISSN: 0098-5589. DOI: 10.1109/TSE.1980.230492 (cit. on p. 1).

[16] Nicolas Treves et al. "A primer on the Petri Net Markup Language and ISO/IEC 15909-2". In: *10th International workshop on Practical Use of Colored Petri Nets and the CPN Tools (CPN'09)*. Aarhus, Denmark, Oct. 2009, p. 19. URL: https://hal.archives-ouvertes.fr/hal-01126017 (cit. on pp. 2, 27, 34).

[17] Karsten Wolf. "Running LoLA 2.0 in a Model Checking Competition". 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. 274–285. ISBN: 978-3-662-53401-4. DOI: 10.1007/978-3-662-53401-4_13. URL: https://doi.org/10.1007/978-3-662-53401-4_13 (cit. on p. 29).

A Unfolding Comparison

Net	verifypn	MCC	Difference	%
AirplaneLD-COL-0010	0.00073	0.00051	0.000226	30.706%
AirplaneLD-COL-0200	0.01201	0.00651	0.005504	45.805%
AirplaneLD-COL-0500	0.03385	0.02389	0.009962	29.428%
AirplaneLD-COL-1000	0.06999	0.04190	0.028095	40.138%
AirplaneLD-COL-2000	0.16769	0.10651	0.061185	36.485%
AirplaneLD-COL-4000	0.42408	0.21614	0.207936	49.032%
BART-COL-002	OOM	OOM		
BART-COL-005	OOM	OOM		
BART-COL-010	OOM	OOM		
BART-COL-020	OOM	OOM		
BART-COL-030	OOM	OOM		
BART-COL-040	OOM	OOM		
BART-COL-050	OOM	OOM		
BART-COL-060	OOM	OOM		
BridgeAndVehicles-COL-V04P05N02	0.00105	0.00057	0.000475	45.238%
BridgeAndVehicles-COL-V20P20N10	0.02765	0.01667	0.010987	39.725%
BridgeAndVehicles-COL-V20P20N20	0.05613	0.03084	0.025287	45.048%
BridgeAndVehicles-COL-V20P20N50	0.13125	0.08463	0.046619	35.517%
BridgeAndVehicles-COL-V50P20N10	0.14524	0.09989	0.045349	31.222%
BridgeAndVehicles-COL-V50P20N20	0.27502	0.19083	0.084193	30.612%
BridgeAndVehicles-COL-V50P20N50	0.62559	0.49256	0.133033	21.264%
BridgeAndVehicles-COL-V50P50N10	0.14457	0.10346	0.041105	28.432%
BridgeAndVehicles-COL-V50P50N20	0.26829	0.18959	0.078698	29.333%
BridgeAndVehicles-COL-V50P50N50	0.69692	0.47386	0.223060	32.006%
BridgeAndVehicles-COL-V80P20N10	0.34855	0.25047	0.098081	28.139%
BridgeAndVehicles-COL-V80P20N20	0.69017	0.47552	0.214650	31.100%
BridgeAndVehicles-COL-V80P20N50	1.53692	1.28707	0.249847	16.256%
BridgeAndVehicles-COL-V80P50N10	0.36159	0.24280	0.118787	32.851%
BridgeAndVehicles-COL-V80P50N20	0.66770	0.48343	0.184269	27.597%
BridgeAndVehicles-COL-V80P50N50	1.66897	1.11859	0.550380	32.977%
CSRepetitions-COL-02	0.00020	0.00013	0.000067	32.682%
CSRepetitions-COL-05	0.00243	0.00112	0.001301	53.539%
CSRepetitions-COL-07	0.00641	0.00280	0.003614	56.336%
CSRepetitions-COL-10	0.01940	0.00782	0.011580	59.681%
DatabaseWithMutex-COL-02	0.00039	0.00024	0.000147	37.404%
DatabaseWithMutex-COL-20	0.05946	0.04038	0.019082	32.088%
DatabaseWithMutex-COL-40	0.37914	0.36747	0.011667	3.0771%

DotAndBoxes-COL-2	0.00611	0.00330	0.002811	45.976%
DotAndBoxes-COL-5	0.34224	0.25558	0.086652	25.319%
DrinkVendingMachine-COL-02	0.00120	0.00074	0.000456	37.968%
DrinkVendingMachine-COL-10	2.45578	1.80053	0.655250	26.681%
DrinkVendingMachine-COL-16	26.672	21.3651	5.30683	19.896%
DrinkVendingMachine-COL-98	OOM	OOM		
GlobalResAllocation-COL-03	0.08269	0.05608	0.026604	32.172%
GlobalResAllocation-COL-05	0.95484	0.80816	0.146676	15.361%
GlobalResAllocation-COL-06	2.73159	2.09632	0.635270	23.256%
GlobalResAllocation-COL-07	6.40236	4.82136	1.58099	24.693%
GlobalResAllocation-COL-09	20.4616	18.4905	1.97108	9.6330%
GlobalResAllocation-COL-10	37.4426	35.2797	2.16284	5.7764%
GlobalResAllocation-COL-11	61.8118	58.0303	3.78144	6.1176%
LamportFastMutEx-COL-2	0.00107	0.00050	0.000568	52.837%
LamportFastMutEx-COL-7	0.00538	0.00252	0.002863	53.146%
LamportFastMutEx-COL-8	0.00614	0.00297	0.003174	51.618%
NeoElection-COL-2	0.00727	0.00382	0.003443	47.352%
NeoElection-COL-6	0.16814	0.12794	0.040206	23.911%
NeoElection-COL-7	0.29943	0.17360	0.125828	42.021%
NeoElection-COL-8	0.41422	0.36415	0.050065	12.086%
PermAdmissibility-COL-01	0.00893	0.00496	0.003970	44.456%
PermAdmissibility-COL-02	0.00912	0.00515	0.003970	43.521%
PermAdmissibility-COL-05	0.0096	0.00381	0.005789	60.302%
PermAdmissibility-COL-10	0.00891	0.00383	0.005085	57.025%
PermAdmissibility-COL-20	0.00928	0.00524	0.004043	43.543%
PermAdmissibility-COL-50	0.00867	0.00381	0.004867	56.084%
Peterson-COL-2	0.00159	0.00100	0.000585	36.792%
Peterson-COL-4	0.00732	0.00317	0.004154	56.702%
Peterson-COL-5	0.01406	0.00551	0.008550	60.780%
Peterson-COL-6	0.02286	0.00881	0.014045	61.433%
Peterson-COL-7	0.03241	0.01782	0.014592	45.014%
Philosophers-COL-000005	0.00020	0.00014	0.000062	30.693%
Philosophers-COL-005000	0.31244	0.17233	0.140112	44.843%
Philosophers-COL-010000	0.82894	0.29570	0.533247	64.328%
Philosophers-COL-050000	11.7449	2.32112	9.42377	80.237%
Philosophers-COL-100000	44.343	4.18525	40.1577	90.561%
PhilosophersDyn-COL-03	0.00123	0.00119	0.000034	2.7619%
PhilosophersDyn-COL-20	0.24562	0.20888	0.036743	14.958%
PhilosophersDyn-COL-50	4.21086	4.25142	-0.04056	-0.9633%
PhilosophersDyn-COL-80	18.3078	16.4749	1.83285	10.011%
PolyORBLF-COL-S02J04T06	0.00871	0.00927	-0.00055	-6.392%

PolyORBLF-COL-S02J06T08	0.01456	0.01210	0.002455	16.858%
PolyORBLF-COL-S02J06T10	0.01711	0.01457	0.002543	14.859%
PolyORBLF-COL-S04J04T06	0.03697	0.03138	0.005587	15.111%
PolyORBLF-COL-S04J04T08	0.05493	0.04086	0.014074	25.618%
PolyORBLF-COL-S04J04T10	0.07476	0.05699	0.017770	23.766%
PolyORBLF-COL-S04J06T06	0.04621	0.03243	0.013773	29.804%
PolyORBLF-COL-S04J06T08	0.05737	0.03962	0.017755	30.943%
PolyORBLF-COL-S04J06T10	0.07947	0.05613	0.023340	29.368%
PolyORBLF-COL-S06J04T04	0.12514	0.07965	0.045498	36.355%
PolyORBLF-COL-S06J04T06	0.18780	0.14773	0.040071	21.336%
PolyORBLF-COL-S06J04T08	0.20636	0.21192	-0.00556	-2.696%
PolyORBLF-COL-S06J06T04	0.12854	0.09955	0.028995	22.555%
PolyORBLF-COL-S06J06T06	0.19380	0.15149	0.042312	21.831%
PolyORBLF-COL-S06J06T08	0.25163	0.21229	0.039338	15.633%
PolyORBNT-COL-S05J20	0.01751	0.01159	0.005916	33.778%
PolyORBNT-COL-S05J30	0.02038	0.01311	0.007270	35.663%
PolyORBNT-COL-S05J40	0.02358	0.01500	0.008586	36.398%
PolyORBNT-COL-S05J60	0.02899	0.02109	0.007901	27.249%
PolyORBNT-COL-S05J80	0.03172	0.02482	0.006905	21.763%
PolyORBNT-COL-S10J20	0.18572	0.18001	0.005704	3.0712%
PolyORBNT-COL-S10J30	0.23997	0.19038	0.049591	20.665%
PolyORBNT-COL-S10J40	0.24193	0.18533	0.056600	23.394%
PolyORBNT-COL-S10J60	0.22964	0.20411	0.025529	11.116%
PolyORBNT-COL-S10J80	0.27839	0.20243	0.075955	27.283%
QuasiCertifProtocol-COL-02	0.00057	0.00042	0.000148	25.919%
QuasiCertifProtocol-COL-10	0.00337	0.00202	0.001344	39.857%
QuasiCertifProtocol-COL-18	0.00752	0.00530	0.002225	29.556%
QuasiCertifProtocol-COL-22	0.01017	0.00945	0.000724	7.1161%
QuasiCertifProtocol-COL-28	0.02128	0.01843	0.002849	13.384%
QuasiCertifProtocol-COL-32	0.02391	0.02058	0.003338	13.956%
Referendum-COL-0010	0.00018	0.00008	0.000100	55.555%
Referendum-COL-0015	0.00025	0.00011	0.000146	56.370%
Referendum-COL-0020	0.00033	0.00014	0.000193	57.100%
Referendum-COL-0050	0.0007	0.00031	0.000382	54.571%
Referendum-COL-0100	0.00131	0.00063	0.000680	51.789%
Referendum-COL-0200	0.00289	0.00100	0.001886	65.191%
Referendum-COL-0500	0.00693	0.00279	0.004132	59.624%
Referendum-COL-1000	0.01568	0.00466	0.011025	70.285%
SafeBus-COL-03	0.00143	0.00075	0.000681	47.522%
SafeBus-COL-10	0.03426	0.01438	0.019877	58.011%
SafeBus-COL-15	0.12741	0.06447	0.062938	49.398%

SafeBus-COL-20	0.34414	0.17652	0.167620	48.706%
SafeBus-COL-50	10.4017	5.58800	4.81369	46.277%
SafeBus-COL-80	58.9255	29.4219	29.5035	50.069%
SharedMemory-COL-000005	0.00056	0.00029	0.000267	47.173%
SharedMemory-COL-000100	0.27429	0.14654	0.127748	46.573%
SharedMemory-COL-000200	2.07812	0.64052	1.43759	69.177%
SharedMemory-COL-050000	TO	OOM		
SharedMemory-COL-100000	TO	OOM		
SimpleLoadBal-COL-02	0.00117	0.00104	0.000129	10.978%
SimpleLoadBal-COL-15	0.04058	0.02572	0.014858	36.608%
SimpleLoadBal-COL-20	0.08017	0.05927	0.020903	26.071%
TokenRing-COL-005	0.00205	0.00144	0.000609	29.620%
TokenRing-COL-040	0.85988	0.70193	0.157949	18.368%
TokenRing-COL-050	1.60725	1.56312	0.044125	2.7453%
TokenRing-COL-100	13.9285	12.6642	1.26427	9.0768%
TokenRing-COL-200	115.51	88.5252	26.9847	23.361%
TokenRing-COL-500	OOM	OOM		
Average	3.6799103	2.5799877	1.0999179	29.890%

B Run time comparison

Net	verifypn	MCC	Difference	%
AirplaneLD-COL-0010	0.04	0.23	-0.19	-475.00%
AirplaneLD-COL-0200	0.07	0.50	-0.43	-614.28%
AirplaneLD-COL-0500	0.14	2.50	-2.36	-1685.7%
AirplaneLD-COL-1000	0.45	14.80	-14.35	-3188.8%
AirplaneLD-COL-2000	1.33	63.51	-62.18	-4675.1%
AirplaneLD-COL-4000	5.06	326.36	-321.30	-6349.8%
BART-COL-002	OOM	OOM		
BART-COL-005	OOM	OOM		
BART-COL-010	OOM	OOM		
BART-COL-020	OOM	OOM		
BART-COL-030	OOM	OOM		
BART-COL-040	OOM	OOM		
BART-COL-050	OOM	OOM		
BART-COL-060	OOM	OOM		
BridgeAndVehicles-COL-V04P05N02	0.00	0.01	-0.01	
BridgeAndVehicles-COL-V20P20N10	0.03	0.06	-0.03	-100.00%
BridgeAndVehicles-COL-V20P20N20	0.06	0.12	-0.06	-100.00%
BridgeAndVehicles-COL-V20P20N50	0.20	0.52	-0.32	-160.00%
BridgeAndVehicles-COL-V50P20N10	0.20	0.33	-0.13	-65.000%
BridgeAndVehicles-COL-V50P20N20	0.34	0.65	-0.31	-91.176%
BridgeAndVehicles-COL-V50P20N50	0.67	2.78	-2.11	-314.92%
BridgeAndVehicles-COL-V50P50N10	0.16	0.32	-0.16	-100.00%
BridgeAndVehicles-COL-V50P50N20	0.28	0.58	-0.30	-107.14%
BridgeAndVehicles-COL-V50P50N50	0.74	2.96	-2.22	-300.00%
BridgeAndVehicles-COL-V80P20N10	0.36	0.61	-0.25	-69.444%
BridgeAndVehicles-COL-V80P20N20	0.76	1.78	-1.02	-134.21%
BridgeAndVehicles-COL-V80P20N50	1.64	10.78	-9.14	-557.31%
BridgeAndVehicles-COL-V80P50N10	0.43	0.60	-0.17	-39.534%
BridgeAndVehicles-COL-V80P50N20	0.74	1.57	-0.83	-112.16%
BridgeAndVehicles-COL-V80P50N50	1.80	9.21	-7.41	-411.66%
CSRepetitions-COL-02	0.04	0.01	0.03	75.0000%
CSRepetitions-COL-05	0.00	0.02	-0.02	
CSRepetitions-COL-07	0.01	0.07	-0.06	-600.00%
CSRepetitions-COL-10	0.04	0.47	-0.43	-1075.0%
DatabaseWithMutex-COL-02	0.00	0.01	-0.01	
DatabaseWithMutex-COL-20	0.15	1.99	-1.84	-1226.6%
DatabaseWithMutex-COL-40	1.57	49.18	-47.61	-3032.4%

DotAndBoxes-COL-2	0.01	0.03	-0.02	-200.00%
DotAndBoxes-COL-5	0.46	27.17	-26.71	-5806.5%
DrinkVendingMachine-COL-02	0.00	0.01	-0.01	
DrinkVendingMachine-COL-10	3.54	440.25	-436.71	-12336%
DrinkVendingMachine-COL-16	38.39	OOM		
DrinkVendingMachine-COL-98	OOM	OOM		
GlobalResAllocation-COL-03	0.12	2.82	-2.70	-2250.0%
GlobalResAllocation-COL-05	1.51	218.29	-216.78	-14356%
GlobalResAllocation-COL-06	4.31	541.24	-536.93	-12457%
GlobalResAllocation-COL-07	10.31	TO		
GlobalResAllocation-COL-09	34.96	OOM		
GlobalResAllocation-COL-10	61.64	OOM		
GlobalResAllocation-COL-11	96.66	OOM		
LamportFastMutEx-COL-2	0.00	0.01	-0.01	
LamportFastMutEx-COL-7	0.01	0.04	-0.03	-300.00%
LamportFastMutEx-COL-8	0.01	0.05	-0.04	-400.00%
NeoElection-COL-2	0.01	0.04	-0.03	-300.00%
NeoElection-COL-6	0.36	14.65	-14.29	-3969.4%
NeoElection-COL-7	0.69	52.09	-51.40	-7449.2%
NeoElection-COL-8	1.04	148.18	-147.14	-14148.%
PermAdmissibility-COL-01	0.02	0.10	-0.08	-400.00%
PermAdmissibility-COL-02	0.02	0.10	-0.08	-400.00%
PermAdmissibility-COL-05	0.02	0.08	-0.06	-300.00%
PermAdmissibility-COL-10	0.01	0.08	-0.07	-700.00%
PermAdmissibility-COL-20	0.02	0.10	-0.08	-400.00%
PermAdmissibility-COL-50	0.01	0.09	-0.08	-800.00%
Peterson-COL-2	0.00	0.02	-0.02	
Peterson-COL-4	0.01	0.06	-0.05	-500.00%
Peterson-COL-5	0.03	0.13	-0.10	-333.33%
Peterson-COL-6	0.05	0.37	-0.32	-640.00%
Peterson-COL-7	0.08	0.84	-0.76	-950.00%
Philosophers-COL-000005	0.00	0.01	-0.01	
Philosophers-COL-005000	4.13	217.08	-212.95	-5156.1%
Philosophers-COL-010000	16.08	171.68	-155.60	-967.66%
Philosophers-COL-050000	373.16	TO		
Philosophers-COL-100000	TO	TO		
PhilosophersDyn-COL-03	0.00	0.02	-0.02	
PhilosophersDyn-COL-20	0.39	47.16	-46.77	-11992%
PhilosophersDyn-COL-50	6.79	TO		
PhilosophersDyn-COL-80	29.07	OOM		
PolyORBLF-COL-S02J04T06	0.02	0.13	-0.11	-550.00%

PolyORBLF-COL-S02J06T08	0.03	0.21	-0.18	-600.00%
PolyORBLF-COL-S02J06T10	0.04	0.30	-0.26	-650.00%
PolyORBLF-COL-S04J04T06	0.06	1.04	-0.98	-1633.3%
PolyORBLF-COL-S04J04T08	0.09	2.04	-1.95	-2166.6%
PolyORBLF-COL-S04J04T10	0.13	3.19	-3.06	-2353.8%
PolyORBLF-COL-S04J06T06	0.08	1.27	-1.19	-1487.5%
PolyORBLF-COL-S04J06T08	0.10	2.19	-2.09	-2090.0%
PolyORBLF-COL-S04J06T10	0.13	3.95	-3.82	-2938.4%
PolyORBLF-COL-S06J04T04	0.18	8.57	-8.39	-4661.1%
PolyORBLF-COL-S06J04T06	0.30	20.02	-19.72	-6573.3%
PolyORBLF-COL-S06J04T08	0.32	31.55	-31.23	-9759.3%
PolyORBLF-COL-S06J06T04	0.21	11.84	-11.63	-5538.0%
PolyORBLF-COL-S06J06T06	0.31	24.40	-24.09	-7770.9%
PolyORBLF-COL-S06J06T08	0.40	29.73	-29.33	-7332.5%
PolyORBNT-COL-S05J20	0.03	1.21	-1.18	-3933.3%
PolyORBNT-COL-S05J30	0.03	1.26	-1.23	-4100.0%
PolyORBNT-COL-S05J40	0.04	0.99	-0.95	-2375.0%
PolyORBNT-COL-S05J60	0.05	1.34	-1.29	-2580.0%
PolyORBNT-COL-S05J80	0.06	1.41	-1.35	-2250.0%
PolyORBNT-COL-S10J20	0.28	21.31	-21.03	-7510.7%
PolyORBNT-COL-S10J30	0.37	37.28	-36.91	-9975.6%
PolyORBNT-COL-S10J40	0.38	34.79	-34.41	-9055.2%
PolyORBNT-COL-S10J60	0.37	28.42	-28.05	-7581.0%
PolyORBNT-COL-S10J80	0.46	33.00	-32.54	-7073.9%
QuasiCertifProtocol-COL-02	0.00	0.50	-0.50	
QuasiCertifProtocol-COL-10	0.01	0.58	-0.57	-5700.0%
QuasiCertifProtocol-COL-18	0.03	0.48	-0.45	-1500.0%
QuasiCertifProtocol-COL-22	0.04	0.48	-0.44	-1100.0%
QuasiCertifProtocol-COL-28	0.09	0.54	-0.45	-500.00%
QuasiCertifProtocol-COL-32	0.14	0.56	-0.42	-300.00%
Referendum-COL-0010	0.00	0.44	-0.44	
Referendum-COL-0015	0.00	0.05	-0.05	
Referendum-COL-0020	0.00	0.63	-0.63	
Referendum-COL-0050	0.00	0.04	-0.04	
Referendum-COL-0100	0.00	0.02	-0.02	
Referendum-COL-0200	0.01	0.64	-0.63	-6300.0%
Referendum-COL-0500	0.03	0.16	-0.13	-433.33%
Referendum-COL-1000	0.08	0.78	-0.70	-875.00%
SafeBus-COL-03	0.00	0.58	-0.58	
SafeBus-COL-10	0.05	0.23	-0.18	-360.00%
SafeBus-COL-15	0.16	2.62	-2.46	-1537.5%

SafeBus-COL-20	0.44	16.51	-16.07	-3652.2%
SafeBus-COL-50	11.92	1105.57	-1093.6	-9174.9%
SafeBus-COL-80	64.81	TO		
SharedMemory-COL-000005	0.00	0.01	-0.01	
SharedMemory-COL-000100	1.00	60.92	-59.92	-5992.0%
SharedMemory-COL-000200	11.63	313.32	-301.69	-2594.0%
SharedMemory-COL-050000	TO	OOM		
SharedMemory-COL-100000	TO	OOM		
SimpleLoadBal-COL-02	0.00	0.02	-0.02	
SimpleLoadBal-COL-15	0.05	0.19	-0.14	-280.00%
SimpleLoadBal-COL-20	0.10	0.52	-0.42	-420.00%
TokenRing-COL-005	0.00	0.01	-0.01	
TokenRing-COL-040	1.27	137.74	-136.47	-10745%
TokenRing-COL-050	2.29	526.31	-524.02	-22882%
TokenRing-COL-100	20.90	TO		
TokenRing-COL-200	180.94	OOM		
TokenRing-COL-500	OOM	OOM		