Bindings in Colored Petri Nets

by

© Sadegh Ekrami, M.Sc.

A thesis submitted to the

School of Graduate Studies

in partial fulfillment of the

requirements for the degree of

Master of Science

Department of Computer Science

Memorial University of Newfoundland

August 2014

St. John's Newfoundland Canada

Abstract

Performance analysis of systems is an important part of system evaluation. If the analyzed system exists, performance analysis can be based on system's measurements (using some sort of instrumentation). If the analyzed system does not exist (as is the case of system upgrading, improvement or design), the approach is to build a (mathematical) model of the system and to use this model for performance analysis. For systems exhibiting concurrency, resource sharing or synchronization of activities, Petri nets are very often used as the modeling formalism. In colored Petri nets, one of nontrivial tasks is to find bindings, i.e. mapping of free variables used in arc expressions to specific colors. Bindings are needed to determine state transitions of a system, therefore, are needed in all analyses of system's behavior. A heuristic approach is proposed which enhances the efficiency of finding bindings in colored Petri nets. Also, performance analysis is used to compare the proposed approach with some other approaches to finding bindings and some remarkable improvements are shown through this analysis.

Acknowledgments

I would like to sincerely thank my supervisor Dr. Wlodek Zuberek for his great help, support and thoughtfulness throughout my program.

I am grateful to School of Graduate Studies and to the Department of Computer Science for financial support. The Natural Sciences and Engineering Research Council of Canada (NSERC) deserves some credit for his financial funding during my time at work on this project.

Also, I should thank Nolan White for his technical assistance and special thanks to my wife, Momeneh Taban, for her love, kindness and support toward my project.

Finally, I want to thank my parents for their love and moral support.

Contents

Abstract **iii**

Acknowledgments **iv**

1 Introduction **3**

1.1 Related Work 5

1.2 Notation 6

1.3 Outline 7

2 Petri Nets **8**

2.1 Petri Nets 8

2.2 Colored Petri Nets 11

2.2.1 Colored Markings 13

2.2.2 Colored Nets 14

2.2.3 Bindings 15

2.3 Summary 16

3 Finding Bindings in Colored Petri Nets **17**

3.1 Potential Transitions 17

3.2 The “Brute Force” Approach 18

3.3 Partial Bindings 20

3.4 The Proposed Approach 23

3.5 Summary 27

4 Performance Analysis of the Proposed Approach 28

4.1 Model Analysis 28

4.1.1 Example 1: Dinning Philosophers 29

4.1.2 Example 2: Multithreaded Shared Memory System 34

4.1.3 Example 3: Extreme Example 37

4.2 Summary 39

5 Concluding Remarks 40

5.1 Implementation 40

5.2 Future Work 41

Bibliography 43

Chapter 1

Introduction

Petri nets are a powerful graphical formalism for the modeling and analysis of systems, especially for systems which exhibit synchronization and concurrency [1, 2, 3]. There are several classes of Petri nets with different properties and different applications. The basic version of Petri nets is known as place/transition nets. The three main components of such nets are places, which represent conditions of a system (drawn as circle or ellipses), transitions, which represent the events (drawn as bars or rectangles), and arcs, which connect places to transitions and transitions to places. This basic model can be extended in several different ways. Petri nets with time [4, 5] include the duration of modeled activities. Colored Petri nets [6, 7, 8, 9] provide a simple representation of complex Petri nets by eliminating replication of similar parts. In colored Petri nets tokens have attributes (called colors), and these attributes can represent a wide range of information associated with tokens. In particular, colors can be used to "fold" similar components of large models (e.g., individual processors of a multiprocessor system) with colors indicating specific components and simplify the structure of the models.

In colored Petri nets, the process of checking if transitions are enabled is more sophisticated than in basic Petri nets because it may involve some relationships between colors of tokens in the input (and output) places. For example, a token with attribute "*a*" in one place may require two tokens with the same attribute in another input place for a transition to fire. Such relationships are described by so called arc expressions. Usually arc expressions are parameterized by using free variables, so they describe the same relationships for different colors. To fire a transition then, all free variables must be associated with specific colors. Such an association of colors to free variables is called binding. An efficient method of finding all possible bindings in colored Petri nets is an important aspect of any approach to the analysis of colored nets.

This thesis proposes a heuristic approach to finding bindings in a colored Petri net model. The proposed appraoch is based on an observation that most arc expressions are very simple, so an efficient handling of such expressions can improve the performance of the analysis of colored nets. A performance analysis of some examples is provided to evaluate efficiency of the analysis. A comparison of the performance of the proposed approach with some other approaches is shown for several examples. Also, some other improvements are discussed to simplify the process of finding bindings. The proposed approach has been implemented in Java and uses MySQL database as its data structure.

1.1 Related Work

Some proposed solutions to the binding problem are overviewed in this section. The approach proposed in [10], uses unfolding of colored nets to eliminate arc expressions and to simplify the checking if a transition is enabled. However, the method cannot be used for all colored Petri nets. In [11] a unification technique (the process of finding assignments under which two algebraic terms are equivalent is often referred to as *unification*) is used to calculate enabled transition instances for the algebraic nets (another kind of high-level Petri nets). Some optimization techniques are considered, however some limitations are that variables on input arcs may not be multiset-valued, and all data types must have finite domains. *Stochastic Well-Formed Nets (SWNs)* are discussed in [12] where an optimized model for computation of *firing sets* is proposed. It is also pointed out that the number of transitions that are affected by the firing of a transition is much smaller than the number of the enabled transitions for a given marking. Hence, this technique can reduce the execution time to find the list of enabled transitions. A simulation has been done according to the presented algorithm and performance improvement has been reported for some specific examples. In [13] an efficient data-structures and algorithms are used to improve the performance of a simulator for colored Petri nets. Only disabled transitions are considered which are discovered during the search for an enabled transition, and the locality principle for an occurring transition is used in order to minimize the changes of enabling status of other transitions. A partial binding method as well as some other optimization techniques are used in [14] to find complete bindings. First the partial bindings are determined for each arc expression and then all partial bindings are merged into complete bindings. During the partial binding test, the number of assignments to free variables decreases. This approach is described in greater details in Chapter 3.

1.2 Notation

The notation used in algorithms in this thesis is summarized in an example called Algorithm 0.

Algorithm 0. An algorithm with all notations used in this thesis.

```
$c$ := RED
for all $x$ in X  do
  Print  $x$
end for
$select$ := true
for all $x$ in X while $select$ do
  if $x = c$ then
    Print $x$, “is equal to RED”
    $select$ := false
  else
    Print $x$, “is not equal to RED”
  end if
end for
```

The assignment operator is “:=” and “=” is used as relational equating operator. “**for all** x **in X**” specifies a **for** loop which iterates x over all elements of the set **X**. In Algorithm 0, the result of the first loop is to print all elements of the set **X**. The next **for** loop “**for all** x **in X while** *select*” prints all elements of **X** until an element is equal to **RED** then the *select* becomes **false**, and the **for** loop ends. All **if** statements end with **end if** and all **for** statement end with **end for**.

1.3 Outline

This thesis is organized as follows: Chapter 2 presents the basic concepts of Petri nets and Colored Petri nets. It also formally introduces the concept of binding. Chapter 3 discusses the proposed approach as well as some other approaches to finding the bindings. An evaluation analysis and performance analysis for some examples is given in Chapter 4. Chapter 5 discusses some of the possible extensions, limitation and future works.

Chapter 2

Petri Nets

This chapter introduces some basic concepts of Petri nets and colored Petri nets. It also introduces the notion of bindings in colored Petri nets and the concept of reachable markings for a Petri net.

2.1 Petri Nets

Petri-nets, proposed by Carl A.Petri in 1962 [15], became one of the most popular formalisms for the description of systems in which the communication between components, synchronization of component operations and resource sharing are essential. Graphically Petri nets are bipartite directed graphs, i.e., directed graphs with two types of vertices called places and transitions. These two types of vertices represent (in a very general sense) conditions (places) and events (transitions). For this reason Petri nets are sometimes called condition-event systems. An event can occur only when some conditions are satisfied (e.g., a car can start its engine if the battery is charged and the start button is pressed), the transition modeling such an event is connected with input places representing these conditions. i.e.,“battery is charged” and “start button is pressed” in this example. An occurrence of an event results in some other conditions that become satisfied (e.g., “the engine is running”), so the transition representing this event will also be connected to some (output) places. Occurrences of events are also called *firings* of (the corresponding) transitions.

Formally, a Petri net $\mathcal{N}$ (also called net structure [16]) is defined as a finite directed bipartite graph, $\mathcal{N} = (\mathbf{P},\ \mathbf{T},\ \mathbf{A})$, where $\mathbf{P}$ is a finite set of places, $\mathbf{P} = \{p_1, p_2, ..., p_n\}$, $\mathbf{T}$ is a finite set of transitions, $\mathbf{T} = \{t_1, t_2, ..., t_m\}$, and $\mathbf{A}$ is a set of directed arcs connecting places with transitions and transitions with places, $\mathbf{A} \subseteq (\mathbf{P} \times \mathbf{T}) \cup (\mathbf{T} \times \mathbf{P})$. Sometimes the set of arcs is defined in two parts, $\mathcal{N} = (\mathbf{P}, \mathbf{T}, \mathbf{A}_1, \mathbf{A}_2)$ where $\mathbf{A}_1 \subseteq \mathbf{P} \times \mathbf{T}$ and $\mathbf{A}_2 \subseteq \mathbf{T} \times \mathbf{P}$. It is convenient to define the input and output sets for each place $p \in \mathbf{P}$, and each transition $t \in \mathbf{T}$:

$$Inp(t) = \{p \in \mathbf{P} \mid (p,t) \in \mathbf{A}\}\,, \tag{2.1}$$

$$Inp(p) = \{t \in \mathbf{T} \mid (t,p) \in \mathbf{A}\}\,, \tag{2.2}$$

$$Out(t) = \{p \in \mathbf{P} \mid (t,p) \in \mathbf{A}\}\,, \tag{2.3}$$

$$Out(p) = \{t \in \mathbf{T} \mid (p,t) \in \mathbf{A}\}\,. \tag{2.4}$$

The dynamic behavior of net models is represented by the so called *tokens* which are associated with places of nets. Any assignment of tokens to places is called a marking function, $m : \mathbf{P} \rightarrow \{0, 1, ..\}$, or simply a marking in a net $\mathcal{N}$. For each

$p \in \mathbf{P}$, $m(p)$ specifies the number of indistinguishable tokens assigned to place p by m. A marked net $\mathcal{M}$ is defined as a pair $\mathcal{M} = (\mathcal{N}, m_0)$ where $\mathcal{N}$ is a Petri net and m_0 is the initial marking function, $m_0 : \mathbf{P} \to \{0, 1, ...\}$. In a marked net $\mathcal{M}$, a transition t is enabled by a marking m if and only if its all input places contain at least one token, i.e.,

$$\forall\, p \in Inp(t) : m(p) > 0. \tag{2.5}$$

If a transition is enabled, an event represented by this transition can occur (i.e., the transition can fire), and such occurrence changes the marking function m into the marking m' for which:

$$\forall p \in \mathbf{P} : m'(p) = \begin{cases} m(p) + 1, & \text{if } p \in Out(t) - Inp(t); \\ m(p) - 1, & \text{if } p \in Inp(t) - Out(t); \\ m(p), & \text{otherwise.} \end{cases} \tag{2.6}$$

It is said that m' is *directly reachable* from m by firing transition t, in notation: $m \xrightarrow{t} m'$. A marking m_j is said to be *generally reachable* from marking m_i in $\mathcal{M}$, notation $m_i \xrightarrow{*} m_j$, if it is reachable from m_i by a sequence of directly reachable markings. The set $\mathbf{M}(\mathcal{M})$ of reachable markings of a marked net $\mathcal{M}$ is the set of all markings that are generally reachable from the initial marking m_0 (including m_0):

$$\mathbf{M}(\mathcal{M}) = \{m \mid m_0 \xrightarrow{*} m\}. \tag{2.7}$$

The *reachability graph* (or the graph of reachable markings) of $\mathcal{M}$, $\mathcal{R}(\mathcal{M})$, a directed, arc-labeled graph $\mathcal{R}(\mathcal{M}) = (\mathbf{V}, \mathbf{E},\ l)$, in which vertices are reachable markings, $\mathbf{V} = \mathbf{M}(\mathcal{M})$, arcs represent the direct reachability relation, $(m_i,\ m_j) \in \mathbf{E} \Leftrightarrow m_i \xrightarrow{t} m_j$, and l is a labeling function which assigns subsets of transitions to elements of $\mathbf{E}$, $l(m_i,\ m_j) = \{t \in \mathbf{T} \mid m_i \xrightarrow{t} m_j\}$.

A marked net $\mathcal{M}$ is *bounded* if there exist a positive number of k such that:

$$\forall\, m\, \forall\, p:\ m(p) \leq k. \tag{2.8}$$

k is called the bound of $\mathcal{M}$. If the bound is equal to 1, the net is called *safe* [1].

It is straightforward to observe that if the set $\mathbf{M}(\mathcal{M})$ is finite, the net is bounded; if $\mathbf{M}(\mathcal{M})$ is infinite, the net is unbounded.

Petri net properties such as *boundedness* [17], *liveness* [18], *reversibility* or *home state* [19], *persistence* [20] and other can be used to check some characteristics of the modeled systems [21, 22]. For example, liveness is closely related to the absence of deadlock, an important aspect of operating systems [23].

2.2 Colored Petri Nets

In colored Petri nets, tokens have attributes (called colors) and these attributes can represent a wide range of information associated with tokens (such as integers for representing number of items in a store or warehouse). In particular, colors can be used to "fold" similar components of large models (e.g., individual processors of a multiprocessor system) where colors indicate specific components and simplifying the structure of the models. However, the analysis of colored Petri nets is much more complex than that of ordinary Petri nets. In Figure 2.2 the three subnets representing the processors in Figure 2.1 are "folded" into one subnet, while the processors are represented by three colored tokens which are quite independent from each other with the exception of accessing memory. If more than one processor accesses memory at the same time, the original priorities are taken into account - the details are discussed in Chapter 4.

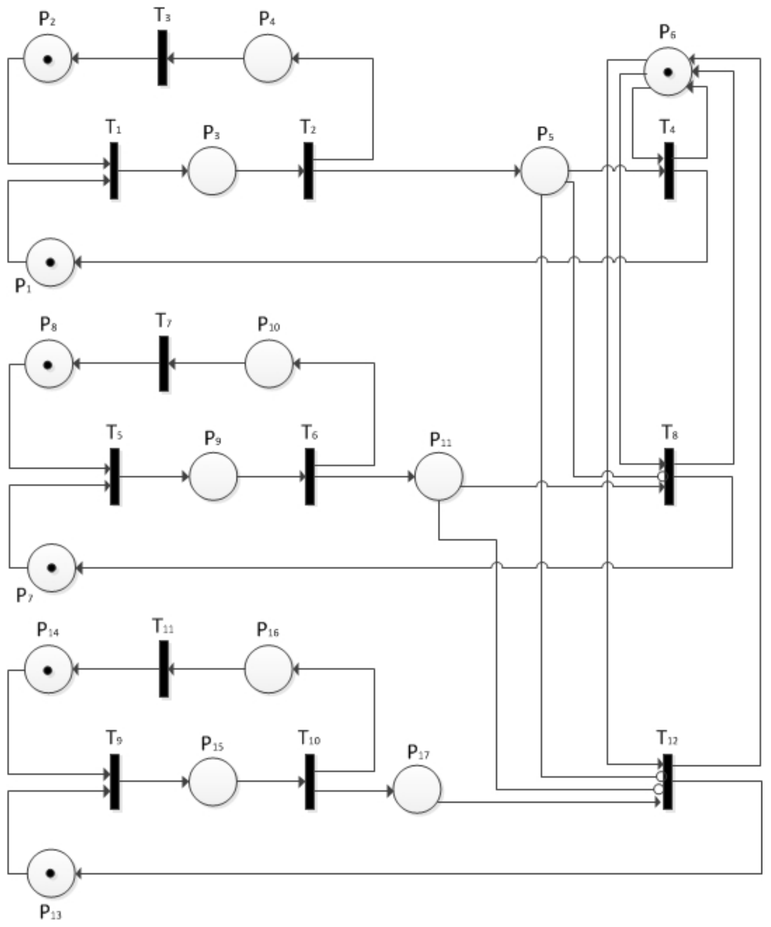

Figure 2.1: The regular Petri net model of three shared memory system.

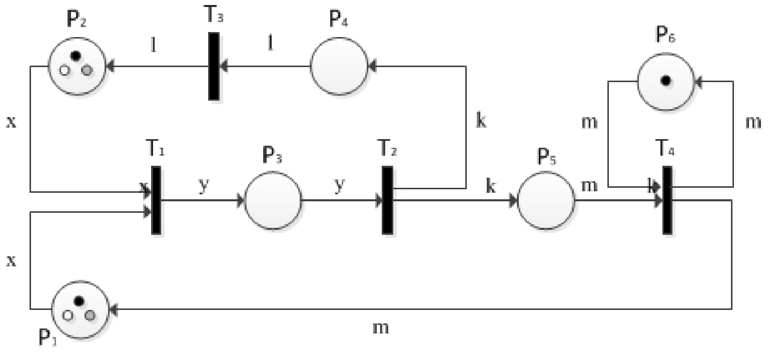

Figure 2.2: The colored Petri net model of three shared memory system.

2.2.1 Colored Markings

In a colored Petri net, a marking of a place is a collection of colored tokens, which is conveniently represented by a function $a : \mathbf{C} \to \{0, 1, ...\}$, often called a bag. A bag is a generalization of a set that allows multiple occurrences of the same element.

An alternative representation of bags, assuming some ordering of colors in the set $\mathbf{C}$, is in the form of vectors:

$$[n_1, n_2, ...] \tag{2.9}$$

where $n_i = g(c_i), i = 1, 2, ...$

Moreover, there is one more popular notation for bags, in which the nonzero elements are written in a form of (linear) expression, for example, if $\mathbf{C}$=$\{blue, green, black, red\}$ and if $m(p)$ is:

$$\forall\ c \in \mathbf{C} : m(p) = \begin{cases} 1, & \text{if } c = red; \\ 2, & \text{if } c \in \{blue, green\}; \\ 0, & \text{otherwise.} \end{cases} \tag{2.10}$$

Then this marking can be written as: $1red + 2blue + 2green$.

Similarly, in colored nets a marking function m is a mapping:

$$\mathbf{C} : \mathbf{P} \to \mathbf{C} \to \{0, 1, ...\} \tag{2.11}$$

i.e., it assigns a bag of colored tokens to each place p in $\mathbf{P}$ (the associativity of the mapping operator $\to$ is from right to left, so $A \to B \to C$ is the same as $A \to (B \to C)$).

2.2.2 Colored Nets

There are several slightly different definitions of colored Petri nets in the literature [6, 9], however, the basic concepts are the same for all these definitions. For this thesis, a colored Petri net $\mathcal{N}$ is defined as $\mathcal{N} = (\mathbf{P}, \mathbf{T}, \mathbf{A}, \mathbf{C}, \mathbf{V}, a)$ where: **(P, T, A)** is a Petri-net structure; **C** is a set of attributes called colors; **V** is a set of free variables; a is an arc function which assigns bags of colors to arcs of $\mathcal{N}$, $a : \mathbf{A} \rightarrow (\mathbf{C} \cup \mathbf{V}) \rightarrow \{0, 1, ...\}$; these bags are in the form of expressions labeling the arcs and they determine the numbers and specific colors of tokens which are used for firing the transitions.

The definition of colored Petri nets, with arc expressions, should be accompanied by some explanations. The arc expressions are actually parameterized bags of colors, with free variables used as parameters. Free variables provide a mechanism of using the same combination of tokens for different colors. There is an implied process of association of free variables with specific colors which is called binding. Formally, binding is a mapping of free variables to colors. A convention used in colored Petri nets is that each binding is restricted to a single transition, i.e., the meaning of free variables in all expressions of arcs incident with the same transition is the same, but the same free variable in arc expressions associated with different transitions may have different bindings. A transition is enabled only if there is a complete binding, i.e., all free variables can be assigned to colors.

2.2.3 Bindings

In colored Petri nets, a transition t is enabled by a marking m if there is such an assignment of colors to free variables in arc expressions associated with t, that the numbers of (colored) tokens in input places of t are not smaller than the values of corresponding arc expressions, i.e., there is a function $b : \mathbf{V}(t) \to \mathbf{C}$ such that:

$$\forall\, p \in Inp(t) : m(p) \geqslant subst(a(p,t), b) \tag{2.12}$$

where $subst(a(p,t), b)$ is the expression associated with arc (p,t) in which free variables are replaced by the values of b.

For example, in Figure 2.3, $m(p_i) = 2white + 1black + 1gray$, and $m(p_j) = 2black + 2white + 2gray$. For this marking, there are two possible bindings. (1) $x = white$, $y = black$, $z = gray$ and (2) $x = white$, $y = gray$, $z = black$. After t's firing, the marking of p_k becomes $1white + 1gray$ for the first binding, or $1white + 1black$ for the second binding. Therefore, the transition t is enabled and can fire with one of these two bindings and these are two different markings reachable from m.

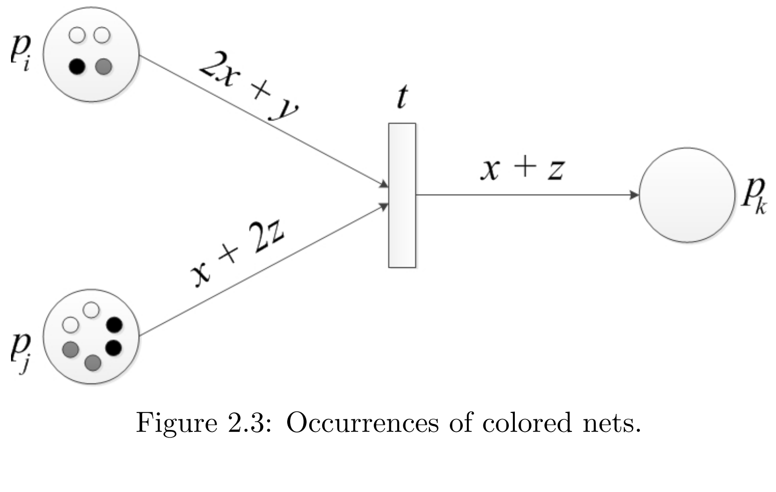

Figure 2.3: Occurrences of colored nets.

A marking m' is reachable from m by firing t with binding b if:

$\forall\ p \in \mathbf{P}:\ m'(p) =$

$$\begin{cases} m(p) + \ subst(a(t,p),b), & \text{if } p \in Inp(t) - Out(p); \\ m(p) - \ subst(a(p,t),b), & \text{if } p \in Out(t) - Inp(p); \\ m(p) + subst(a(t,p),b) - \ subst(a(p,t),b), & \text{if } p \in Inp(t) \cap Out(p); \\ m(p) & \text{otherwise.} \end{cases} \tag{2.13}$$

2.3 Summary

This chapter has introduced the basic concepts of ordinary Petri nets and colored Petri nets (CPNs). Formal definitions have been provided and bindings in colored Petri nets also have been discussed.

Chapter 3

Finding Bindings in Colored Petri Nets

An efficient method of finding bindings in colored Petri nets is essential for any behavioral analysis of a colored net. An overview of approaches to finding all possible bindings is presented in this chapter. It is assumed that predicates (or conditions associated with transition) are transformed to arc expressions, so they are not considered explicitly.

3.1 Potential Transitions

Finding bindings in colored net is a rather intricate process, which can be made more efficient by eliminating steps which cannot be successful. Selecting potential transitions eliminates - from the further process - all transitions which cannot be enabled. For a marking m, the set of *potential transitions*, (denoted as $\mathbf{T}_P(m)$), contains those transitions whose input places are all marked and colors of the marked places can be assigned to their arc expressions, so:

$$t \in \mathbf{T}_P(m) \;\Leftrightarrow\; \forall\, p \in Inp(t)\; \exists\, c \in \mathbf{C}(p) :\; nzn(m(p)) \geq card(\mathbf{V}(a(p,t))) \qquad (3.1)$$

where $\mathbf{C}(p)$ is the set of colors of tokens assigned to p by m, $nzn(m(p))$ is the number of nonzero elements in bag $m(p)$ and $card(\mathbf{V}(a(p,t)))$ is the number of free variables in the arc expression $a(p,t)$.

In other words, if an input place of a transition is not marked, then there is no need to check this transition for bindings. Also, places which are marked but they do not have enough colored tokens to satisfy the binding assignment of their arc expressions are not good candidates to be considered for the *enabling test*.

Algorithm 1 outlines the procedure of finding the set of potential transitions.

Algorithm 1 Finding potential transitions for marking m.

```
$\mathbf{T}_P(m) := \emptyset$
for all $t$ in $\mathbf{T}$ do
    $check$ := true
    for all $p$ in $Inp(t)$ while $check$ do
        if $nzn(m(p)) < card(\mathbf{V}(a(p,t)))$ then
            $check$ := false
        end if
    end for
    if $check$ then
        $\mathbf{T}_P(m) := \mathbf{T}_P(m) \cup \{t\}$
    end if
end for
```

3.2 The "Brute Force" Approach

For a given marking m, the "brute force" approach systematically checks all possible assignments of colors to free variables in order to find the bindings for which a transition is enabled. So, if the set of free variables for a transition t is $\mathbf{V}(t) = \{v_1, ..., v_k\}$, the approach can be as follows:

Algorithm 2 Brute force algorithm to find possible bindings for the transition t at marking m.

```
for all $v_1$ in $\mathbf{C}(m,t)$ do
  for all $v_2$ in $\mathbf{C}(m,t)$ do
    if $v_2 \neq v_1$ then
      for all $v_3$ in $\mathbf{C}(m,t)$ do
        if $v_3 \neq v_2$ and $v_3 \neq v_1$ then
          ...
          for all $v_k$ in $\mathbf{C}(m,t)$ do
            if $v_k \neq v_{k-1}$ and ... and $v_k \neq v_1$ then
              for all $p$ in $Inp(t)$ do
                check if $t$ is enabled by $m$
              end for
            end if
          end for
        end if
      end for
    end if
  end for
end for
```

where $\mathbf{C}(m,t)$ is the set of colors of tokens in input places of transition t for marking m: $\mathbf{C}(m,t) = \bigcup_{p \in Inp(t)} \mathbf{C}(m,p)$ and $\mathbf{C}(m,p)$ is the set of colors of tokens in place p in marking m:

$$\mathbf{C}(m,p) = \{c \ \in \ \mathbf{C} \mid m(p)(c) \ > \ 0\} \tag{3.2}$$

In Algorithm 2, the number of **for** loops corresponds to the number of free variables in all arc expressions associated with transition t. All **if** conditions check whether the color assigned to each variable is not the same as colors assigned to the other variables. This procedure is applied to all potential transitions for a given marking m.

For example, in Figure 2.3, $a(p_j, t)$ is "$x + 2z$" and $a(p_i, t)$ is "$2x + y$". The algorithm will assign $x = black$ and $y = white$ and $z = gray$ in one of its iterations which is an unsuccessful assignment for $a(p_i, t)$. There are five other assignments for this example. Two of the six assignments enable the transition t, i.e. one is $x = white$, $y = black$, $z = gray$ and the other is $x = white$, $y = gray$, $z = black$.

Although this approach finds all possible successful assignments, it can be quite time consuming if several free variables are used. The brute force approach is used in Chapter 4 as the basis for performance comparison with other approaches to finding the bindings. For the performance analysis of this approach the operation counts are considered as the comparison analysis in Chapter 4.

3.3 Partial Bindings

The partial binding approach proposed in [14], first finds a partial binding for each arc expression and then merges the partial bindings into complete bindings. During the partial binding test, the number of assignments to free variables decreases. In fact, each of the variables is processed sequentially and before the verification of other variables. This method processes all expressions specified for the selected transition and finds all valid assignments of colors to variables. This process ends when all free variables have been checked.

The partial binding process assigns all colors of the input place p of the transition t to the variables of arc expression associated with this place and transition.

In [14] the partial binding principle has been used as well as some other techniques for preprocessing and ordering the colored Petri net structure. As mentioned earlier, partial binding sets of a transition need to be *compatible*. Two bindings, b_1 and b_2, are compatible, which is denoted as $\mathit{Compatible}(b_1, b_2)$, when

$$\forall\ v \in \mathbf{V}(t):\ b_1(v) \neq \perp\ \wedge b_2(v) \neq \perp\ \Rightarrow\ b_1(v) = b_2(v). \tag{3.3}$$

Where $\mathbf{V}(t)$ is the set of free variables associated with transition t (i.e., used in all arc expressions incident with t), and $\perp$ indicated that the value is undefined.

A combined binding, denoted as $\mathit{Combine}(b_1, b_2)$, of two compatible partial binding defined as:

$$\forall\ v \in \mathbf{V}(t) : \mathit{Combine}(b_1, b_2)(v) = \begin{cases} b_1(v), & \text{if } b_1(v) \neq \perp \\ b_2(v), & \text{if } b_2(v) \neq \perp \\ \perp, & \text{otherwise.} \end{cases} \tag{3.4}$$

Moreover, merging of two partial binding sets $\mathbf{B}_1$ and $\mathbf{B}_2$, denoted as $Merge(\mathbf{B}_1, \mathbf{B}_2)$, is defined as:

$$Merge(\mathbf{B}_1, \mathbf{B}_2) = \{\mathit{Combine}(b_1, b_2)\ |\ \ b_1 \in \mathbf{B}_1\ \wedge\ b_2 \in \mathbf{B}_2\ \wedge\ \mathit{Compatible}(b_1, b_2)\}. \tag{3.5}$$

Algorithm 3 shows the process of finding bindings and merging partial bindings. This algorithm searches for all input arc expressions associated with transition t and does the process of partial binding test for each color c in the color set $\mathbf{C}(m, p)$ by calling the function $\mathit{PartialBindings}(e, c)$. $\mathbf{C}(m, p)$ contains the colors for place p at marking m. $\mathit{PartialBindings}(e, c)$ performs the successful assignment of colors to variables in the expression e. All partial bindings are stored in $\mathbf{B}'$ and are merged with binding set $\mathbf{B}$. $\mathbf{B}(m, t)$ is the set of bindings for transition t and marking m. $\mathit{PartialBindings}(e, c)$ is the partial binding function with assigns the colors c to the variables of arc expression e and returns a binding.

Algorithm 3 Partial binding algorithm to find possible bindings for the transition t at marking m

```
$\mathbf{B} := \emptyset$
for all $p$ in $Inp(t)$ do
  $\mathbf{B}' := \emptyset$
  $e := a(p,t)$
  for all $c$ in $\mathbf{C}(m,p)$ do
    $\mathbf{B}' := \mathbf{B}' \cup \{PartialBindings(e,c)\}$
  end for
  $\mathbf{B} := Merge(\mathbf{B}, \mathbf{B}')$
end for
```

Figure 3.1 shows a single transition of a colored Petri net. Algorithm 3 is performed for this example to find bindings as follows. To get the complete bindings, first the partial bindings are created for place p_1 with marking $2black+2white+2gray$ and with the arc expression "$2x+2gray$". The *gray* tokens are used by the constant part of the arc expression so the two partial bindings are:

$$b_1 = (x = black, y = \perp).$$

$$b_2 = (x = white, y = \perp).$$

For place p_2 with the arc expression "$x+y$", there are six partial bindings:

$$b'_1 = (x = black, y = white).$$

$$b'_2 = (x = black, y = gray).$$

$$b'_3 = (x = white, y = black).$$

$$b'_4 = (x = white, y = gray).$$

$$b'_5 = (x = gray, y = black).$$

$$b'_6 = (x = gray, y = white).$$

Therefore, the compatible bindings of these two sets of bindings, $\{b_1, b_2\}$ and $\{b'_1, b'_2, b'_3, b'_4, b'_5, b'_6\}$, are: $\mathbf{B}' = \{(x = black, y = white), (x = white, y = black)\}$. The next arc expression is "$x+gray$" with one partial binding because the *gray* token is explicitly required by the arc expression so the free variable can only have the value *black*:

$$b_1'' = (x = black, y = \perp).$$

Consequently, the final merged set of bindings is: $\mathbf{B} = \{(x = black, y = white)\}$.

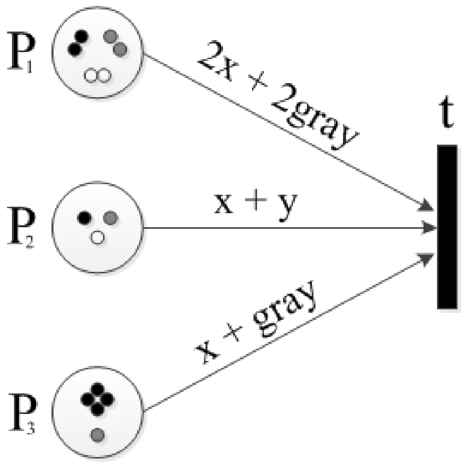

Figure 3.1: A part of a colored Petri net example.

3.4 The Proposed Approach

The proposed approach is based on an observation that many arc expressions are very simple (are linear). If the types of arc expressions (for some nets guard expressions are also considered[10]) are different, the implementation of transition enabling and firing algorithms might change significantly[12]. One of the approaches would be to unify different patterns that can cover all possible arc expressions.

For linear arc expressions, a specialized approach can be used to process such expressions more efficiently than general arc expressions. Therefore, arc expressions are divided into 2 classes:

1) Linear expressions: written as a sum of colors or free variables, possibly associated with an integer coefficient indicating the number of tokens of a respective color:

```
expr  ::= count | expr + count
count ::= color | integer color | var | integer var |
```

A logical function $linear(expr)$ returns true if the argument $expr$ is a linear expression. For example; $linear(2x + y)$ returns true.

2) General expressions (e.g. functions, conditions etc).

Algorithm 4 shows the process of finding all possible assignment of colors to free variables for a transition $t \in \mathbf{T}_P(m)$. As before $\mathbf{V}(t)$ is the set of free variables for transition t. For each variable v in $\mathbf{V}(t)$, place p' is sought whose arc expression contains v ($v \in \mathbf{V}(a(p, t))$). In this way the number of checks whether a color can be assigned to a variable is reduced. Then, for all colors of the places that are associated with the transition t ($c \in \mathbf{C}(m, t)$) and for all places whose arc expressions contain the variable v ($p \in Inp(t, v)$) a checking is performed to investigate valid assignments. If an expression is linear ($linear(a(p, t))$) then this checking is based on the number of colors and the coefficients of the selected variable. Otherwise, the substitution of variables and colors is checked to investigate its validity. For example, in Figure 2.3 $m(p_1)$ is $2white + 1black + 1gray$ and the arc expression $a(p_1, t)$ is "$2x + y$". $m(p_1)(black)$ is 1 and if the color $black$ is assigned to the variable x, the result of this substitution will be $2black + y$. As $subst(a(p_1, t), black)(black)$ is 2, this assignment is unsuccessful. This process ends with a set of the valid assignments of colors to variables in $\mathbf{S}$ as a collection of pairs (variable, color).

Algorithm 4 Finding Bindings for transition t and marking m

```
$\mathbf{S} := \emptyset$
for all $v$ in $\mathbf{V}(t)$ do
  $Success :=$ true
  $First :=$ true
  for all $p \in Inp(t,v)$ do
    if $First$ then
      $First :=$ false
      $p' := p$
    end if
    for all $c$ in $\mathbf{C}(m,p')$ while $Success$ do
      if $linear(a(p',t))$ and $m(p')(c) < num(a(p',t),v)$ then
        $Success :=$ false
      else if $\neg linear(a(p',t))$ and $m(p')(c) < subst(a(p',t),c)(c)$ then
        $Success :=$ false
      end if
      for all $p$ in $Inp(t,v)$ while $Success$ do
        if $linear(a(p,t))$ and $num((a(p,t),v) > m(p)(c)$ then
          $Success :=$ false
        else if $\neg linear(a(p,t))$ and $m(p)(c) < subst(a(p,t),c)(c)$ then
          $Success :=$ false
        end if
      end for
      if $Success$ then
        $\mathbf{S} := \mathbf{S} \cup \{(v,c)\}$
      end if
    end for
  end for
end for
```

In Algorithm 4 it is assumed that all colors of the marking m are ordered with respect to the numbers of tokens assigned by m in descending order; the variables of linear expressions are also ordered based on their coefficients in descending order. This helps to reduce the number of checks performed. $Inp(t,v)$ is the set of input places of t such that arc expressions $a(p,t)$ contain the variable v; $num((a(p,t),v)$ is the coefficient of the variable v in the arc expression $a(p,t)$. $Success$ is used to indicate a successful assignment of a color to a variable. $First$ is used to select the first place whose arc expression contains variable v and which contains sufficient number of tokens. Figure 3.2 illustrates this process for a transition with all arc expressions that are linear.

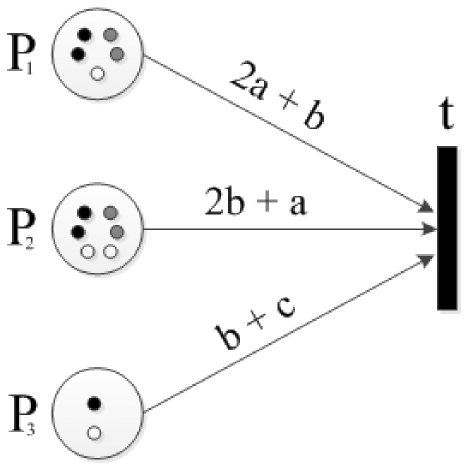

Figure 3.2: A part of a colored Petri net example.

The set of all free variables for transition t is $V(t) = \{a, b, c\}$. Let the first variable be a. The first arc expression which contains this variable is (p_1, t). The colors available in all places are sorted based on the numbers of tokens so that p_1 contains $\{2black, 2gray, 1white\}$, p_2 contains $\{2black, 2gray, 2white\}$ and p_3 contains $\{1black, 1white\}$. A comparison based on the number of each color and the coefficient of variables is made to find possible assignments. This helps to reduce the number of comparisons since the coefficient of a variable might be greater than the number of colored tokens which means that the rest of the color set will not be compared. Then for all places whose arc expressions contain a (i.e. "$2a+b$","$2b+a$"), all possible colors available in all input places will be assigned to the selected variable (which is a for the first iteration). Two colors, *black* and *gray*, could be associated with the variable a. So, at the end of the first iteration over the first variable, the pairs $(a, black)$ and $(a, gray)$ are added to $\mathbf{S}$. The variable b is considered next and the first place whose arc expression contains the variable is selected (p_1 in this example). Again, for all colors in p_1 the algorithm checks for valid assignment. The first color is *black* (as it has sorted). The other arc expression containing variable b is "$b + c$". The first assignment is $(b, black)$, however, b can not be assigned to *gray* as there is no *gray* in p_3. The added sets to the $\mathbf{S}$ after the second iteration are $(b, black)$ and $(b, white)$. Consequently, after the last iteration $\mathbf{S} = \{(a, black), (a, gray), (b, black), (b, white), (c, black), (c, white)\}$).

In this example two bindings are created in $\mathbf{S}$. b_1 is: $a = gray$, $b = black$ and $c = white$. b_2 is $a = gray$, $b = white$ and $c = black$. For the performance analysis of this approach the operation counts are considered as the comparison analysis in Chapter 4.

3.5 Summary

In this chapter, the proposed approach for finding enabled transitions and their binding(s) has been discussed. The counts of basic steps are introduced which are used in chapter 4 for comparisons of different approaches.

Chapter 4

Performance Analysis of the Proposed Approach

This chapter compares the properties of the proposed approach with some other ones. Several examples (small and large models) are used to compare the performance of finding possible bindings and new markings. The analysis is performed by software in the Java programming language combined with databases for the data structures of colored Petri nets.

4.1 Model Analysis

All algorithms for finding bindings in colored Petri nets depend on many factors such as the size of net models, the number of enabled transitions at each marking, complexity of arc expressions and many other elements which might have significant effects on analysis efficiency. Several popular examples were used to check the influence of some factors that affect the efficiency of the analysis of colored nets.

4.1.1 Example 1: Dinning Philosophers

The dinning philosopher example is one of popular examples used to illustrate synchronization and concurrency of processes. It has been proposed by Dijkstra[24]. As is known, philosophers follow a cycle of two operations, thinking and dinning. Their favorite food is spaghetti and to eat spaghetti in a proper way two forks are needed. However, there are only five forks, so, the philosophers share the forks at a round dinning table and only those philosophers who have two forks can eat their spaghetti. Usually the forks are placed (on the dinning table) between philosophers, so each philosopher has its right and left fork. Figure 4.1 shows a colored net model of this example.

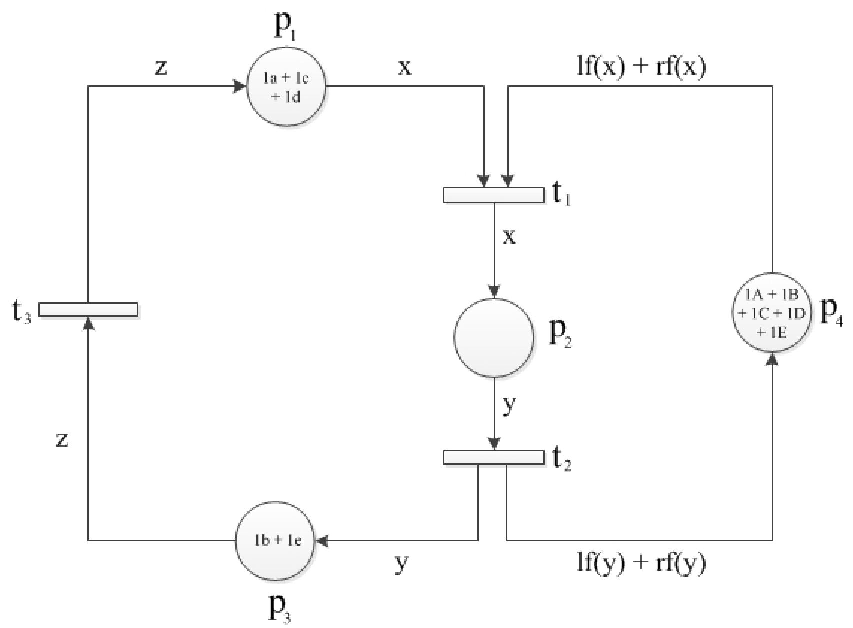

Figure 4.1: Colored Petri net model of five dinning philosophers.

In this example philosophers are represented by colors a, b, c, d and e. The forks are represented by colors A, B, C, D and E. The two functions, $lf(x)$ and $rf(x)$ assign the left and right fork to each philosopher x, so, as shown in Table 4.1, $lf(a) = A$, $rf(a) = B$, $lf(b) = B$, $rf(b) = C$, and so on; i.e., fork A is shared by philosophers a and e, fork B by philosophers a and b, and so on.

Table 4.1: Left and right forks for each philosophers.

Philosopher x	lf(x)	rf(x)
a	A	B
b	B	C
c	C	D
d	D	E
e	E	A

In Figure 4.1, place p_4 represents available (i.e. not used) forks. Place p_1 represents philosophers who are waiting to get their forks and eat, place p_2 philosophers who are eating and place p_3 philosophers who are thinking. The initial marking in Figure 4.1 represents a situation where all forks are available, philosophers b and e are thinking, while the remaining three philosophers (a, c and d) are ready to eat.

Transition t_1 corresponds to fetching the two forks in order to start eating. The free variable x, representing a philosopher, is used in the functions $lf(x)$ and $rf(x)$ which associate the forks with each philosopher. For the initial marking shown in Figure 4.1, there are three possibilities of firing t_1, with x corresponding to a, or to c or to d. Consequently, there are 3 next marking functions created by firing t_1 in m_0.

Transition t_2 represents end of eating and depositing the left and right forks to p_4, and beginning of the thinking phase.

Transition t_3 correspond to the end of thinking and readiness for eating (if the forks are available). For the marking shown in Figure 4.1, there are two possible firing of t_3, one corresponding to philosopher b, and the other to philosopher e.

For the initial marking showed in Figure 4.1, there are 5 next markings, each of which creates a number of subsequence markings, and so on. The net shown in Figure 4.1, has 152 reachable markings (including the initial marking).

The number of markings for different number of philosophers (from 3 to 7) is shown in Figure 4.2. It can be easily shown that the number of markings grows exponentially with the number of philosophers.

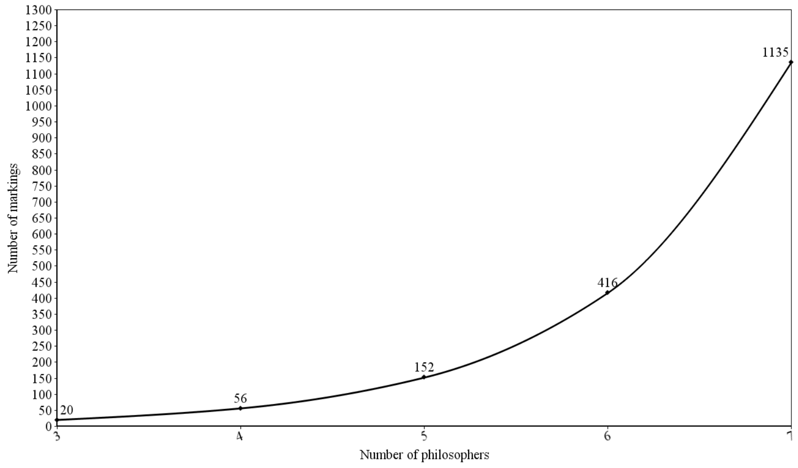

Figure 4.2: The number of reachable markings for dinning philosophers as a function of the number of philosophers.

To compare the performance of different approaches to finding the bindings, the number of required common steps can be used. The checking if a transition is enabled is used as the basis of performance comparison. Tables 4.2 and 4.3 show the number of checks for the brute force approach and the proposed approach.

Table 4.2: The number of checks for finding possible bindings for marking m_0 with brute force approach and the proposed approach.

For m_0	Number of Checks
Brute force approach	8
Proposed approach	5

Table 4.3: The number of checks for finding possible bindings for all markings with brute force approach and the proposed approach.

For all markings	Number of Checks
Brute force approach	780
Proposed approach	620

The arc expressions in this example are quite simple so the number of checks in brute-force approach is fairly low. However, the efficiency of the proposed approach is showing an improvement in the whole number of checks over the running of the program to find all possible markings of the net.

An analysis for different number of philosophers is performed in Figure 4.3 to show the trend of the total number of checks as a function of the number of dinning philosophers.

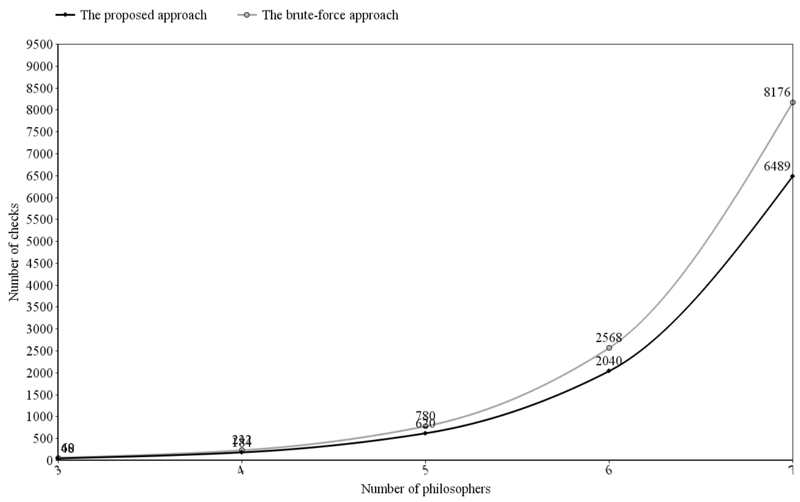

Figure 4.3: The performance of the proposed approach and the brute-force approach as a function of the number of dinning philosopher.

As it can be seen in Figure 4.3, the number of checks for the brute-force approach is growing similarly to the number of checks for the proposed approach.

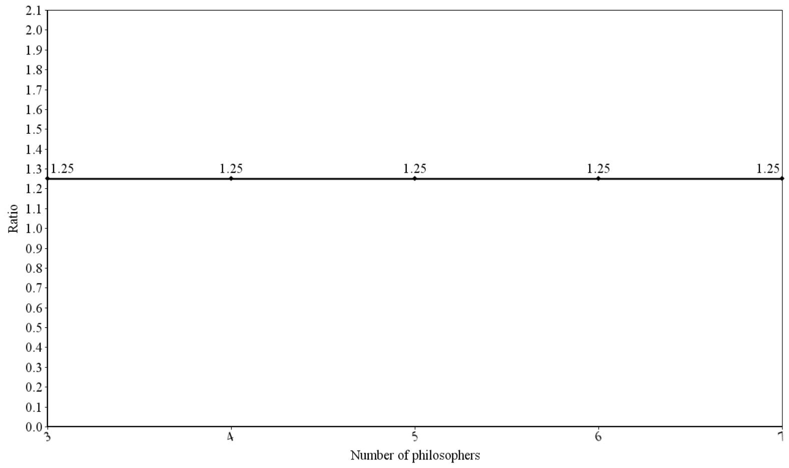

Figure 4.4: The ratio of performance of the brute-force approach to the proposed approach as a function of the number of philosophers.

Figure 4.4 shows the ratio of these two number of checks (the brute-force approach over the proposed approach) as a function of the number of philosophers.

4.1.2 Example 2: Multithreaded Shared Memory System

In this example, several multithreaded processors are connected to a shared memory, so only one of these processors can access the memory at any time. Figure 4.5, shows a colored Petri net model of multithreaded shared memory system with four multithreaded processors.

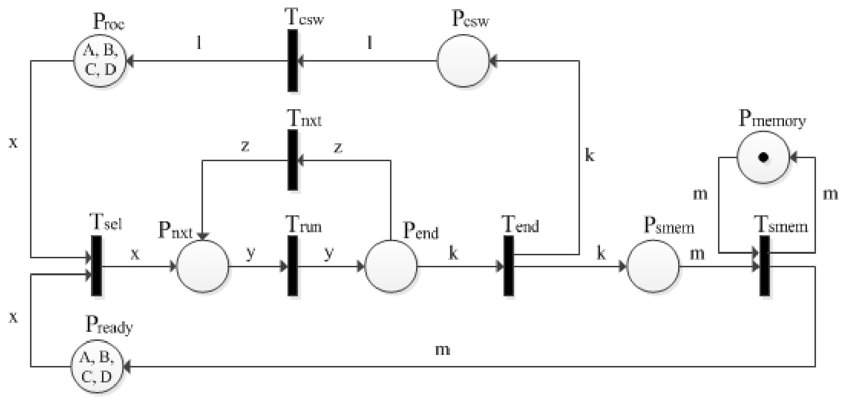

Figure 4.5: Colored Petri net model of shared memory system with four processors.

Processors are represented by colors A, B, C and D. Initially all processors are in the place P_{roc}. If there are available threads in P_{ready}, the execution can begin by firing T_{sel}. Execution of consecutive instructions of the selected thread is performed in the loop P_{nxt}, T_{run}, P_{end} and T_{nxt}. If T_{end} is chosen for firing rather than T_{nxt} the execution of the thread ends and a request to access the shared memory is placed in P_{smem} where it waits for availability of local memory by P_{memory}. Also, a token is deposited in P_{csw} to perform context switching by firing T_{csw}.

Accessing shared memory (T_{smem}) is controlled by priorities of processors. If a processor with higher priority requires the memory then processors with lower priorities cannot access the memory until the processor with higher priority finishes its memory access. Table 4.4 shows the priority list of this multithreaded shared memory example. The highest priority is denoted by 1, the lowest by 4. Therefore, the processor A has higher priority than processors B, C and D. The processor B has higher priority than processors C and D and so on.

Table 4.4: Priority list of processors.

processor	Priority
A	1
B	2
C	3
D	4

For the initial marking showed in Figure 4.5, there are 4 next markings, each of which creates a number of subsequence markings, and so on. For the net shown in this figure, there are 1296 reachable markings (including the initial marking).

The number of markings for different numbers of processors (from 2 to 5) is shown in Figure 4.6. It can be shown that the number of markings grows exponentially as a function of the number of processors.

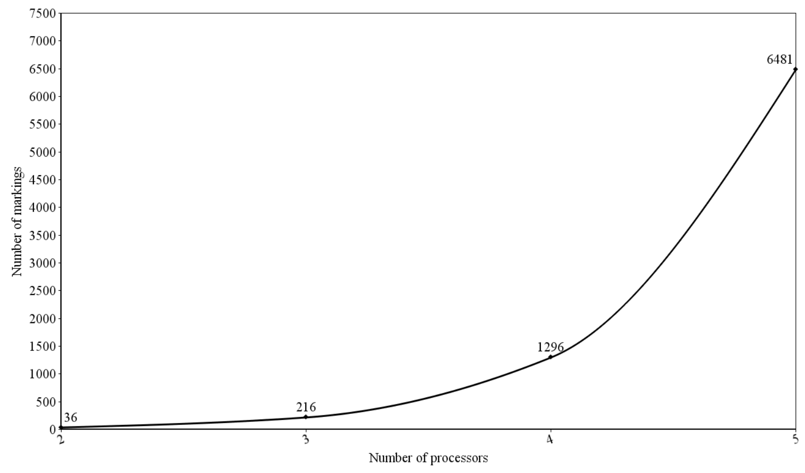

Figure 4.6: The number of reachable markings for multithreaded shared memory as a function of the number of processors.

The performance for different number of processors is shown in Figure 4.7.

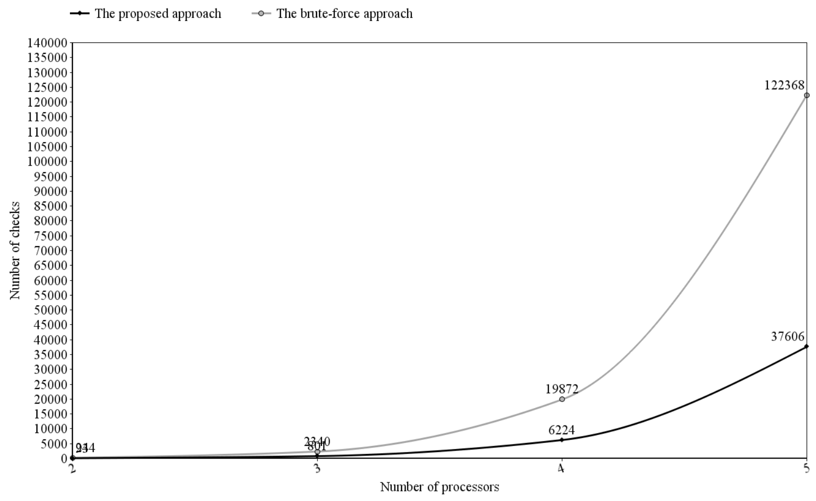

Figure 4.7: The performance of the proposed approach and the brute-force approach as a function of the number of processors.

As it can be seen in Figure 4.7, the number of checks required by the brute-force approach is significantly greater than that for the proposed approach. Figure 4.8 shows the ratio of these two numbers of checks (for the brute-force approach to the proposed approach) as a function of the number of processors. The results indicate that the performance of the proposed method actually improves as the models become more complex, although more evidence is needed supporting this observation.

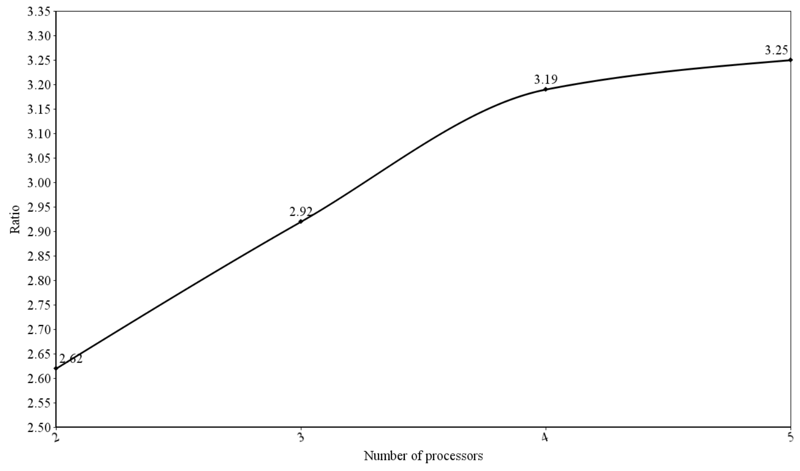

Figure 4.8: The ratio of performance of the brute-force approach to the proposed approach as a function of the number of processors.

4.1.3 Example 3: Extreme Example

In the previous examples the differences between the two approaches were rather limited because of the simple arc expressions as well as simple input places associated with transitions. Therefore, to show the differences between these approaches, an extreme example has been created only with a single transition. Figure 4.9 shows this special example.

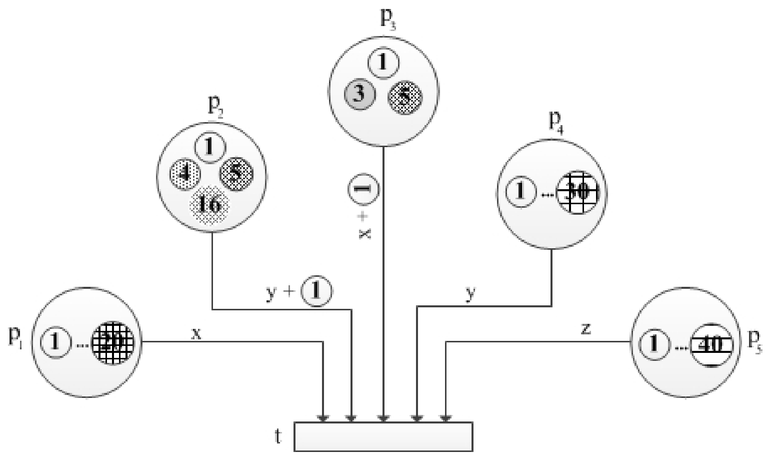

Figure 4.9: Colored Petri net model of an extreme example for one transition.

There are 5 different input places associated with transition t. There are 40 different colors in $\mathbf{C}(t)$. p_1 contains 20 tokens with colors denoted 1 to 20. p_2 contains four different colors denoted 1, 4, 5 and 16. p_3 contains three different colors denoted 1, 3 and 5. p_4 contains 30 different colors denoted 1 to 30 and p_5 contains 40 different colors denoted 1 to 40. There is only 1 token of each color all places.

Performance analysis has been performed by three different approaches described in Chapter 3, i.e. the brute force approach, the partial binding approach and the proposed approach. In Table 4.5 the best result of each approach is shown. As it can be clearly seen the number of marking produced for this example is 152 and the result of the proposed approach is much better than the brute-force approach and almost 3 times better than the partial binding approach.

Table 4.5: Performance analysis of the special extreme case with three different approaches

Approach	Number of bindings	Number of checks
Proposed approach		240
Partial binding	152	605
Brute-force		20880

4.2 Summary

The performance of several net models has been compared. These results indicate that practical implementations of the proposed approach is showing an improvement of the proposed approach with the other approaches for complex cases.

Chapter 5

Concluding Remarks

A heuristic approach for finding bindings in colored Petri nets is proposed. In this thesis the results of performance analysis presented in Chapter 4 show that the gains of the proposed approach are more significant for larger and more complex nets. In this thesis, the performance evaluation is based on the numbers of steps needed for finding the bindings. However, consideration of execution time for different approaches may be more relevant for colored net models. The execution time is mostly dependent on the implementation techniques, the interpretation of the net models and the Petri net structure [25]. It is expected that these ideas will be further developed in future work.

5.1 Implementation

All performance results were obtained by a simple net analyzer implemented in the Java programing language combined with the MySQL database. A colored Petri net structure is defined in the MySQL database and the Java program simply uses the information of a net to perform enabling test for finding bindings and creating new markings. The approaches are developed following the algorithms presented in Chapter 3.

5.2 Future Work

As mentioned earlier, Petri net structures have a significant effect on the efficiency of analysis. *Unfolding* is one of the approaches which transform a colored Petri net model into a regular Petri net model. As the specification of regular Petri nets is simpler than colored Petri nets, the process of finding all markings is more straightforward. So, an analysis of the unfolding approach versus the presented approaches is one of the possible future projects. This helps to estimate how a net structure can affect the overhead of the unfolding process. Also, the performance analysis of the approaches according to the net structure will be considered. Net unfolding, however, can be a challenging task.

Unbounded colored net models are challenging because the unfolding process is quite impossible. In order to make the proposed approach working for some unbounded net models, similar and repetitive markings should be stored and compared with other markings to prevent repetitive process of finding bindings. This can help to reduce the overhead of enabling test process. If similar markings (similar patterns) can have same effect for enabling test then the pattern matching of markings could help to find bindings easier. *Memoization* is one of the possible techniques which can be used to save time at the expense of space. This technique stores the results corresponding to some set of specific inputs [26]. This can help to reduce the overhead of calling a function (i.e. firing of a transition and producing markings) based on inputs that have been calculated before.

Moreover, as the analysis presented in Chapter 4 are all based on number of the computational steps, a better comparison can be based on the execution time specifically for the binding process (not for the whole part of the program, as it may cause other effects on execution time). Execution time is depended on many parameters such as the net structure, the implementation techniques and also different interpretation of net models (i.e. how inputs, actions and code are associated with the net elements). So, a standard implementation should be considered to estimate and analyze the execution time accurately.

Bibliography

[1] T. Murata, “Petri Nets: Properties, Analysis and Applications,” *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[2] W. Reisig, *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer Verlag, 2013.

[3] W. Reisig, *Petri Nets: An Introduction*, vol. 4 of *EATCS Monographs in Computer Science*. Springer-Verlag, 1985.

[4] W. M. Zuberek, “Timed Petri Nets and Preliminary Performance Evaluation,” in *Proceedings of the 7th Annual Symposium on Computer Architecture*, (ISCA ’80), pp. 88–96, ACM, 1980.

[5] J. Wang, *Timed Petri Nets - Theory and Application*, vol. 9 of *The International Series on Discrete Event Dynamic Systems*. Kluwer Academic Publisher, 1998.

[6] K. Jensen, “Coloured Petri Nets: A High Level Language for System Design and Analysis,” *Advances in Petri Nets 1990; Lecture Notes in Computer Science*, vol. 483, pp. 342–416, Springer-Verlag, 1991.

[7] K. Jensen, "An Introduction to the Theoretical Aspects of Coloured Petri Nets," in *A Decade of Concurrency, Lecture Notes in Computer Science*, pp. 230–272, Springer-Verlag, 1994.

[8] K. Jensen, *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*, vol. 1 of *An EATCS Series. Monographs in Theoretical Computer Science.* Springer, 1996.

[9] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems," *IEEE Trans. Inf. Theor.*, vol. 22, pp. 213–254, Sept. 2006.

[10] M. Sanders, "Efficient Computation of Enabled Transition Bindings in High-Level Petri Nets," in *2000 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, pp. 3153–3158, 2000.

[11] M. Makela, "Optimising Enabling Tests and Unfoldings of Algebraic System Nets," in *International Conference on Application and Theory of Petri Nets, Lecture Notes in Computer Science*, pp. 283–302, Springer, 2001.

[12] R. Gaeta, "Efficient Discrete-Event Simulation of Colored Petri Nets," *IEEE Trans. Software Eng.*, vol. 22, no. 9, pp. 629–639, 1996.

[13] K. Mortensen, *"Efficient Data-Structures and Algorithms for a Coloured Petri Nets Simulator"*, pp. 57–75. DAIMI PB, Department of Computer Science, Aarhus University, 2001.

[14] F. Liu and M. Heiner, "Computation of Enabled Transition Instances for Colored Petri Nets," in *Proc. 17th German Workshop on Algorithms and Tools for Petri Nets*, CEUR Workshop Proceedings, vol. 643, pp. 51–65, 2010.

[15] C. A. Petri, *Kommunikation mit Automaten.* PhD thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.

[16] W. M. Zuberek, "Petri Nets and Timed Petri Nets - Basic Concepts and Properties," *Technical Report, Department of Computer Science, Memorial University*, December 2001.

[17] E. Pastor, J. Cortadella, and O. Roig, "Symbolic Analysis of Bounded Petri Nets," *IEEE Trans. Comput.*, vol. 50, pp. 432–448, May 2001.

[18] E. Kindler and W. van der Aalst, "Liveness, Fairness, and Recurrence in Petri Nets," *Inf. Process. Lett.*, vol. 70, no. 6, pp. 269–274, 1999.

[19] T. Miyamoto and S. Kumagai, "Reversibility Verification of Petri Nets Using Unfoldings," in *1997 IEEE International Conference on Computational Cybernetics and Simulation in Systems, Man, and Cybernetics*, vol. 5, pp. 4274–4278, 1997.

[20] E. Best and J. Esparza, "Model Checking of Persistent Petri Nets," in *Proceedings of the 5th Workshop on Computer Science Logic*, (CSL '91), pp. 35–52, Springer-Verlag, 1992.

[21] R. Bonnet, "The Reachability Problem for Vector Addition System with One Zero-test," in *Proceedings of the 36th International Conference on Mathematical*

Foundations of Computer Science, (MFCS'11), pp. 145–157, Springer-Verlag, 2011.

[22] P. Abdulla and R. Mayr, "Computing Optimal Coverability Costs in Priced Timed Petri Nets," in *26th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pp. 399–408, 2011.

[23] L. Jiao, "A Method for Verifying Deadlock Freedom and Liveness of Petri Nets," in *IEEE International Symposium on Circuit and System*, pp. 209–211, 2008.

[24] J. Díaz and I. Ramos, "Communicating Sequential Processes," in *International Colloquium in Formalization of Programming Concepts*, vol. 107, pp. vii – 478, Springer-Verlag, 1981.

[25] R. P. Moreno and J. L. V. Salcedo, "Performance Evaluation of Petri Nets Execution Algorithms.," in *System, Man, and Cybernetics*, pp. 1400–1407, IEEE, 2007.

[26] Memoization, "Memoization — Wikipedia, the free encyclopedia," http://en.wikipedia.org/wiki/Memoization 2014. [Online; accessed 11 April 2014 at 10:58].

[27] L. Popova-Zeugmann, *Time and Petri Nets.* Springer Verlag, 2013.

[28] C. Girault and R. Valk, *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications.* Springer-Verlag, 2002.

[29] E. Smith, "Principles of High-Level Net Theory," in *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the Volumes Are Based on the Advanced Course on Petri Nets*, pp. 174–210, Springer-Verlag, 1998.

[30] J. M. Ilié and O. Rojas, "On Well-Formed Nets and Optimizations in Enabling Tests," in *Proceedings of the 14th International Conference on Application and Theory of Petri Nets*, pp. 300–318, Springer-Verlag, 1993.

[31] S. Haddad, F. Kordon, L. Petrucci, J.-F. Pradat-Peyre, and L. Treves, "Efficient State-Based Analysis by Introducing Bags in Petri Nets Color Domains," in *American Control Conference, 2009. ACC '09.*, pp. 5018–5025, 2009.

[32] L. M. Kristensen and A. Valmari, "Finding Stubborn Sets of Coloured Petri Nets Without Unfolding," in *Proceedings of the 19th International Conference on Application and Theory of Petri Nets*, ICATPN '98, pp. 104–123, Springer-Verlag, 1998.

[33] M. Mäkelä, "Optimising Enabling Tests and Unfoldings of Algebraic System Nets," in *Proceedings of the 22Nd International Conference on Application and Theory of Petri Nets*, ICATPN '01, pp. 283–302, Springer-Verlag, 2001.

[34] Y. Xu, X. Xie, D. Xia, Z. Liu, and L. Chen, "Modeling and Analysis of an Online Score System Using Colored Petri Nets," in *Proceedings of the 3rd International Conference on Anti-Counterfeiting, Security, and Identification in Communication*, ASID'09, pp. 432–436, IEEE Press, 2009.