Doctoral Dissertation
(Shinshu University)

On the Formal Verification of Petri Net Properties using a Mechanized Proof Checker Approach

(プルーフチェッカーシステムを用いたペトリネットの性質の形式的検証について)

SEPTEMBER 2014

PRATIMA KUMARI SHAH

Interdisciplinary Graduate School of Science and Technology
Department of Mathematics and System Development
Faculty of Engineering
Shinshu University

Dedication

This doctoral dissertation entitled "**On the Formal Verification of Petri Net Properties using a Mechanized Proof Checker Approach**" is dedicated to my beloved Mom Chanchala Devi Shah and Daddy Prof. Satya Narayan Shah, who introduced me to the magnificent nature of the world, and also to my husband Dr. Shailendra K. Shah and son Nathan Preet Shah, who have been my backbone during this period.

September, 2014

.....

Pratima Kumari Shah
Department of Mathematics and System Development
Faculty of Engineering
Shinshu University, Nagano Campus, Japan

Acknowledgements

I would like to thank all the people who assisted me in completing this work.

First, I would like to express my deep appreciation to my supervisor, Pauline N. Kawamoto, Assoc. Prof., Department of Computer Science and Engineering, Shinshu University, for her support and fruitful discussions throughout my graduate studies. Those discussions pointed me in the right direction, and greatly influenced on my research work. Special thanks to Yasunari Shidama, Professor, and Minoru Maruyama, Professor, Department of Computer Science and Engineering, Shinshu University for being my co-supervisor during my study and also for their helpful advice during writing this dissertation.

I am in debt to all kind people at the Department of Computer Science and Engineering, Shinshu University, all staffs of the International Student Center, Faculty of Engineering have been most helpful with administrative assistance.

I also wish to acknowledge the Japanese Government of Education, Culture, Sports, Science and Technology (Monbukagakusho MEXT) for providing full scholarship that supported my doctoral research study and living costs in Japan.

I have deep gratitude towards my parents, friends and fellow-students for their kindness and support throughout the years of my study. Finally, I want to extend my heartfelt and deepest appreciation to my husband Dr. Shailendra K. Shah, who helped me continuously in my study and my son Nathan Preet Shah for staying in day care center (Hoikuen) without my accompany and sharing pains and pleasures with me during my completion of this study.

Contents

List of Abbreviations

1. CSP- Communicating Sequential Process
2. ATPs- Automated Theorem Provers
3. ITPs- Interactive Theorem Provers
4. PTNs- Petri nets
5. DFPNs- Decision free Petri nets
6. TPs- Theorem Provers.
7. DPs- Directed Paths
8. DCs- Directed Circuits.
9. DEDS- Discrete Event Dynamic Systems
10. FMs- Formal Methods
11. MML- Mizar Mathematical Library
12. HOL- Higher Order Logic
13. LCF- Logic for Computable Functions
14. FOL- First Order Logic

List of Figures

List of Tables

Abstract

In this work, we use a theorem checking system called Mizar to formalize token boundedness property of a subclass of Petri nets. The rigorous Mizar formalization and mechanical theorem verification presented in this work uncovers a common example of how information omitted from natural language descriptions of mathematical concepts and proof can lead to areas of ambiguity.

Both natural language and formal language descriptions of hardware and software systems and their properties are widely used in the area of computer science. Petri nets are often used to mathematically model and examine concurrent behaviors of hardware and software systems. However, when definitions and proofs of Petri net properties are expressed in natural language, details (trivial proof) that appear obvious for expert mathematicians or are repeated often are sometimes omitted from these descriptions, leaving the reader to fill in the gaps in order to understand. This kind of problem can occur because definitions in natural language are not always precise. Some missing information can cause errors or ambiguity in interpretation when developers try to apply these concepts to tools for system modeling and analysis. Such errors may have catastrophic consequence in terms of money and time. In general an earlier an error is detected, the cheaper it is to fix, which eventually produces reliable hardware and software systems with high quality.

In this work, the Mizar system is used for the formal verification of token boundedness property of a subclass of Petri nets called decision free Petri nets (available in the literature [1]), which has the characteristics that every place in the net has exactly one input transition and one output transition, and mechanically verify that this formalization is semantically and logically correct. The Mizar language descriptions are more complete because every statement in this language must be fully justified with no jumps in the chain of logic. Using a number of basic Petri nets constructs already verified by the Mizar system and accepted to its repository of mathematical information, we built an extension for the decision free Petri nets and proved the token boundedness property of their circuits. While formalizing, we found clear examples of how gaps available in natural language descriptions of mathematical concepts and proofs in literature can lead to areas of ambiguity and highlight the value of using mechanical systems proof checking mechanisms to formalize such notions and archive all details of the corrected materials

in the open digital library of Mizar for reuse. This type of archiving of mechanically verified Petri net knowledge can serve as a reliable source for system modeling tool developers who rely on a common and clear understanding of the concept definitions and proofs, and also for the beginner reader.

The motivation of this research is to demonstrate the power of formalization techniques and its application to the formal verification of theorems concerning mathematically rich Petri net models with high reliability. In the future, the formal proofs of decision free Petri nets in the database of the theorem checking system can be used to analyze the liveness and safeness properties in directed circuits.

Chapter 1 INTRODUCTION

This chapter provides an introduction to the research work presented in this dissertation. It describes the research background and explains the purpose of the research work. In addition, it also provides an objective of this work. Finally, it introduces the structure of the dissertation.

1.1 Background

The work presented in this dissertation is the confluence of formal verification, where computer verifies the correctness of the formalized mathematics completed by humans. With the inevitable increase in the complexity of computer hardware and software systems, the likelihood of the level of subtle errors is high. Such errors may have a disastrous event in terms of time and cost [2]. If an error is detected earlier in system development phase, it would be easier and cheaper to fix. The error during the system design can be the result of several reasons, maybe because of omitted information or any ambiguity from given requirements. The requirements in software development process can be mathematical formulas, definitions, and theorems, which mostly found in some textbooks and literatures written in natural language. In the system development area there is a growing demand for methodologies that can fix errors and increase the confidence in correct system design [3]. Such methodologies will result in improved quality of hardware and software systems. The formal verification approach is useful to detect and help the developers to correct errors during the hardware and software development process [3]. The traditional engineering approach to the construction of complex systems is to build models first, which can be studied and modified until the confidence is obtained in their correctness [4]. Formal verification advocated a similar approach to the construction of computing systems. The background information on the formal verification method and its different approaches, including theorem checking system is presented in Appendix A.

1.2 Purpose of the Research Work

Over the years, both natural language and formal language descriptions of Petri nets and their properties are widely used in the area of system development [5]. Petri nets are often used to mathematically model and examine concurrent and synchronous behaviors of hardware and software systems [6]. However, when concepts of Petri nets structures and properties are

expressed in natural language, some trivial proofs that appear obvious for expert mathematicians are sometimes purposely or accidentally omitted from these descriptions. This missing information can cause errors or ambiguity in interpretation when developers try to apply these concepts to proof assistant tool for system modeling and analysis. This kind of problem can occur because definitions written in natural language are not always accurate. This can be a source of much frustration for a beginner reader. The reader has to fill in the gaps of omitted facts and trivial proofs in natural language description in order to understand the exact meaning of concepts. Formal language descriptions on the other hand, are more complete because every statement must be fully justified.

In this work, we used a theorem checking system called Mizar to write mathematical definitions and formalize token boundedness property of decision free Petri nets, where Mizar verifier checks its logical correctness [1], [7]. Using a number of basic Petri net constructs already verified by the Mizar system and accepted to its digital library, we extended the available knowledge of Petri nets in library by building the formalization of decision free Petri net structure. Then we successfully proved the token boundedness property in circuits of decision free Petri net for both transition and transition sequence firing [8], [9]. Boundedness is always necessary to be able to keep the Petri net model of a manufacturing system bounded; for example, an unbounded model may cause buffer overflow [10]. The rigorous Mizar formalization and mechanical theorem verification presented in this work reveals a common example of how information omitted from natural language descriptions of mathematical concepts and proofs can lead to areas of ambiguity and highlight the value of using mechanical proof checking approach to formalize such notions and archive all details of the corrected materials for reuse.

1.3 Gaps in Natural Language Definitions and Theorems

In system development area, an unambiguous mathematical definition is necessary so that we know the exact meaning of what the definition is talking about. Unfortunately, for many of the concepts or expressions in natural language, the definition is rather difficult to understand, so often, beginners use an intuitive feeling for the meaning of an expression. This intuitive feeling, while necessary, sometimes is not accurate and sufficient. This means, we need to overcome with the problem and master the formal statement of definitions and their meaning. For example, if one reads expression like $p_1t_1p_2t_2... p_k$, at first glance, one might, not sure whether the

subscripts in the expression are either the position in the sequence or are part of the element labels.

Theorems are usually the important result which shows how to make concepts/definitions solve problems or give major insight into the working of the system design. Theorems use definitions which must have been given precise, means if definitions is not correct, theorem won't satisfy. So the considerable amount of care must be taken in the statement of the definitions and theorems in making it possible to formalize and prove the theorem completely and correctly [11]. Moreover, for the theorem to be correct, the conclusion should cover all the cases, not just simple or convenient case. Unfortunately, some proof in most literature considers only the most convenient and simple case [1]. For example, if we consider an excerpt of token boundedness proof from literature that "tokens in the circuit can only be produced or consumed by transitions in the circuit, i.e., when a transition consumes a token, it produces back into the circuit, therefore the number of tokens in a circuit remains the same after any firing sequence". While formalizing, we found that this proof does not hold for all cases, i.e., the exact meaning of this proof is only sufficient for the cases when the circuit does not contain repeated elements. In other word, readers after reading such proof and trying to satisfy the proof in a circuit with repeated elements, one might find that the number of tokens in the circuit before and after firing some transitions are not same. The brief description of this gap with diagrams is given in Chapter 2.

1.4 Objectives of the present study

The impetus for the present research came from a growing need for reliable hardware and software systems [3]. Complexity in the hardware and software systems increases the level of errors. For many years, researchers have been using formal verification methods to generate a reliable software and hardware systems. The advantage of using this method is to find system defects as early as possible and eventually cutting the efforts and budgets needed for redesigning any systems [2]. System failure not only results significant time and budget loss, but can also be life hazardous. The recent example of budget loss is the presence of Knight Capital's computer bugs in an automated trading program cost its proprietor 440 million dollars in 30 minutes which took place on August 2012 [12]. The reason of most system failures is because of a lack of guard against unexpected inputs and sometimes also because of mismatch between model specification

and actual software, which could have been prevented by a lot of attention to strict coding as well as thorough reviews which can be partly done by computer and partly by human. To this point, Mizar is a tool where Mizar language is used for writing code by humans and its verifier with the aid of computer verifies the logical correctness of the written code.

Up to now, some foundation of the basic Petri nets structure is formalized and stored in the Mizar mathematical library (MML) [8], [9], [13], [14]. However, there is a need to analyze and store more and more knowledge of the Petri nets structural and behavioral properties to construct and analyze even more complex Petri net model. Consequently, I turned my attention to mechanically build a structure of a subclass of Petri nets on top of the foundation of the Petri nets basic structure present in the Mizar library. After building a subclass of Petri net (decision free Petri net), one of its behavioral properties called token boundedness property is chosen and formalized in Mizar to check if the proof given in the literature is enough and safe to interpret in any modeling tool. As mentioned above, the present information in literature to prove the property was not enough and thus we had to add several definitions, lemmas, and theorems, which was not even talked or considered in the literature.

The major objective of the present study is to use the power of formalization techniques to clarify and strengthen the concepts of the mathematically rich modeling language of Petri nets by:

-- constructing the rigorous and complete descriptions of definitions and proofs using a mathematical language which will not allow for the exclusion of any cases that can cause ambiguity or error in interpretation, and

-- contributing the constructed mathematical knowledge, which has a computer guarantee of logical correctness to an open digital library, where users can easily access to retrieve the required information for the verification of the new complex mathematical proof.

This type of archiving of mechanically verified Petri nets knowledge can serve as a reliable source for system modeling tool developers who rely on a common and clear understanding of the concept definitions and proofs, and also for the beginner readers who are seeking to get a deeper understanding of mathematics.

1.5 Dissertation Outline

In Chapter 2, we discuss the outline of the present research, where we show how omitted information in natural language descriptions can create gaps in the reasoning and why certain analytical properties of Petri nets are actually true.

In Chapter 3, we describe the formalization of decision free Petri nets and its token boundedness property, where we show which omitted cases and definitions were needed to solve the considered property in the Mizar system.

Chapter 4 provides the conclusion and future work of the study presented in this dissertation.

Appendix A contains the background information on the formal verification method and its different approaches.

Appendix B contains the background information on the Petri nets concepts which was used in this work for the formal verification of Petri nets behavioral properties by using mechanical approach.

Appendix C contains the Mizar background and some symbols and notations of Mizar that are used during the formalization of this work.

Appendix D contains the abstract of the list of definitions and theorems that was required to prove token boundedness property.

Chapter 2 OUTLINE OF RESEARCH

This chapter presents an outline of the research work, where we used a theorem checking system to formalize a token boundedness property in a circuit of decision free Petri net structure. Early on, we found that natural language descriptions of mathematical concepts can have ambiguous descriptions that can be interpreted in different ways by different readers. Also, proofs are sometimes incomplete.

In this study, we found cases of both as we tried to build the formal descriptions of concepts related to token boundedness property of decision free Petri net. The background information on the basic concepts of Petri nets written in natural language description is presented in Appendix B. When we discuss about mathematical definitions and theorems, first we will consider definitions because they form the foundation for any part of mathematics and are essential for understanding and proving theorems.

2.1 Gaps in Natural Language Definitions

A definition in mathematics should be a precise statement delineating and naming a concept by relating it to previously defined concepts. When we discuss the boundedness property of decision free Petri nets, two substructures: directed paths and directed circuits become necessary and in this work, this was the first example of how omitted facts in natural language definitions can cause ambiguity in interpreting the exact meaning of mathematical objects [1]. According to the Petri nets directed path definition, a finite alternating sequence of places and transitions, for example, $p_1t_1p_2t_2... p_k$, is a directed path if and only if transition t_i is both an output transition of place p_i and an input transition of place p_{i+1} for $1 \leq i \leq K-1$. Similarly, by Petri net directed circuit definition, a finite alternating sequence of places and transitions, like $p_1t_1p_2t_2...p_k$, is directed circuit if and only if the sequence is directed path, where first and the last element of the sequence are equal [1], [15].

If one follows both definitions of directed path and directed circuit, one might not be clear whether the subscripts in the labels of the sequence elements (e.g., "1" in p_1) represent the positions in the sequence or the subscripts are part of element labels (e.g., a place element called "p_1"). The clear meaning about this confusion on numbering should have discussed in the literature before presenting the definition of directed path and directed circuit.

2.2 Gaps Revealed in Natural Language Proofs of Theorems

Theorems and its proofs when written in natural language description usually admit some ambiguity. Formal proofs, written in formal language must be unambiguous, complete, and must demonstrate that the proof of the theorems is true in all cases, without a single exception. When we interpret the theorem of token boundedness property of decision free Petri nets in Mizar, we found that the given proof of this property in literature is not true for all cases, which was required to fix before proving this particular theorem.

a). Proof of Token Boundedness Property taken from literature [1]:

The available proof of the token boundedness property in literature is completed in about less than 3 lines with a Fig. 2.1, as shown below [1].

"*theorem*: For a decision free Petri net, the number of tokens in a circuit remains the same after any firing sequence.

Proof: Tokens in the circuit can only be produced or consumed by transitions in the circuit. When a transition consumes a token, it produces one back into the circuit; therefore, the number of tokens in a circuit remains the same after any firing sequence "

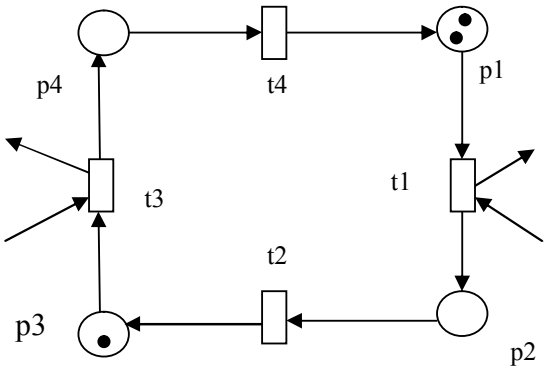

Figure 2.1. Decision free Petri net with circuit

b). Translation of above proof written in natural language to formal language

Suppose if a programmer 'A' considers proof of this property as it is from literature to interpret it in some tool, proof might work for kind of circuit as shown above in Fig 2.1. But

accidentally or purposely, this programmer didn't check the written proof for the circuit shown in Fig. 2.2, and thus Programmer 'A' might not get any error.

Again, if the same proof from literature is used by the programmer 'B' to interpret it in any theorem checker, this programmer found that the token boundedness property is true for a circuit like Fig 2.1. But there is a problem with the case of circuit like Fig. 2.2, where the alternating sequence of places and transitions, say p1, t1, p2, t2, p3, t3, p1, t1, p4, t4, p5, t2, p3, t3, p1 forms a circuit. With circuit like Fig 2.2, containing repeated elements, a simple check will show that they do not preserve the number of tokens after firing some transitions. As an example, the initial marking M0 of Fig 2.2 nets are (1, 0, 0, 0, 0). Then after firing enabled transition t1, the new marking M1 in Fig. 2.2 is (0, 1, 0, 1, 0). The transition firable and firing rule of the Petri net model is discussed in Appendix B. It can be clearly seen that the summation of total number of tokens in circuit before and after firing the transition t1 are not same. Hence the token boundedness property in a circuit that has a duplicate element is not as straightforward as suggested by natural language descriptions of the property in the literature [1]. If the formalization of the token boundedness property was continued without this realization, the work proof of the theorem could not be completed. This is because, in natural language proof, sometimes only the most convenient cases are addressed. But in the theorem prover, all cases must be explicitly proven before it is accepted as true by the verifier of the proof checker.

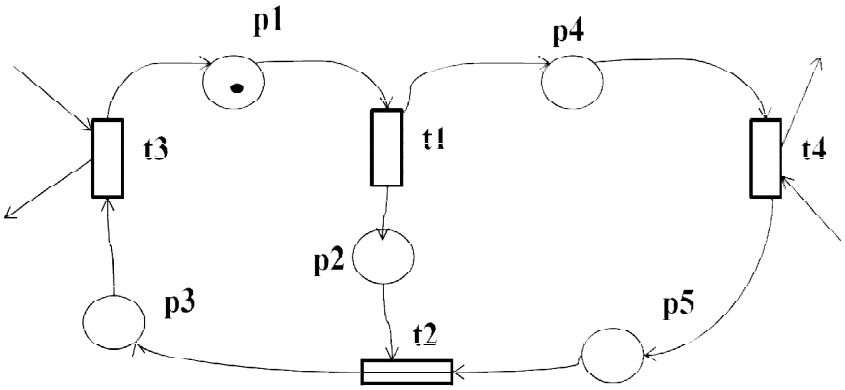

Figure 2.2. Circuit contains repeated elements.

The above explanation concludes that all the cases should have explicitly written in the

literature, so that the programmer could understand clearly before interpreting the exact meaning of theorems and its proof into any tool. But in this case, we could not find such definitions stating that this boundedness property will not hold for the circuit that has repeated elements in literature. In order to fix the errors or in order to consider all the cases to prove this particular property correctly and completely, we had to create a definition sating that the circuits must not contain repeated elements except first and last elements. Such circuit, in general, is called an elementary circuit.

Since formal proof checks every possible case, another major error during the reasoning process was in the case of transition firing outside the circuit (for example, in this case firing of transition t5 in Fig. 2.3). The question arises that what happens when transitions not present in the circuit would fire. Whether it affects the number of tokens in the circuit or not. Logically, it does not change the circuits token number in this subclass of Petri net, but formal proof wants this fact to be verified before verifying the token boundedness theorem.

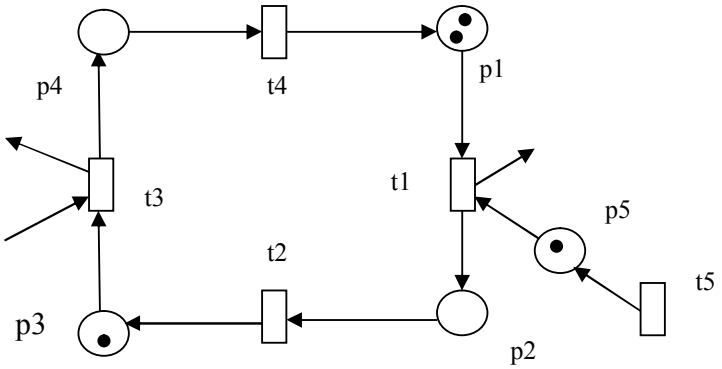

Figure 2.3. Firing of transition outside the circuit

There were several other trivial facts that we had to define thoroughly and prove one by one in the Mizar tool to verify that the token boundedness proof is syntactically and logically correct in all cases [16], [17]. This verification took a couple of thousand lines to prove the boundedness property in a circuit of decision free Petri net, which in literature is just given in even less than 3-line. Finally, by using all these definitions, and several accompanying lemmas and propositions, we proved token boundedness property for both firing a single transition and firing a transition sequence.

2.3 Mizar

Mizar is an advanced project of the Mizar society led by Andrzej Trybulec that formalizes mathematics with a computer-aided proving technique [7], [18]. The Mizar project describes mathematical proofs in the Mizar language, which is created to formally describe mathematics. It has been a popular and efficient tool because of its powerful ability to formalize and analyze a model description and proposition computation. In this paper, we used the Mizar proof checking system because it is the most mathematically-oriented one, adopting a grammar resembling common mathematical language, a declarative style, and being based on first-order predicate logic. The background information on the Mizar System, its standard logical connectives and quantifiers are presented in Appendix C.

Chapter 3 Formalization of Decision Free Petri Net and the Boundedness Theorem

As discussed in the first two chapters that the core aim of this dissertation is to demonstrate a practical and effective approach for analyzing the token boundedness behavioral property of decision free Petri nets by using the Mizar system. Here we tried to keep the derivation of definitions and theorems of Petri nets as close to the textbook as much as possible. Definitions and theorems related to decision free Petri net and its considered property are formalized and divided into six subsections whose abstract form is presented in Appendix D and the complete proof of all these definitions and theorems is available online [17].

3.1 Token Boundedness Theorem

The theorem of the token boundedness property described in natural language in literature is as follows, "for a decision free Petri net, the number of tokens in a circuit remains the same after any transition firing sequence" [1]. The natural language proof of this theorem is presented in Chapter 2. In order to prove the token boundedness theorem for transition firing sequence, first we need to prove the token boundedness property for firing single transition, which is used as one of the references to prove the property for firing transition sequence. Different people can have different ideas for how to translate the same theorem and its proof. Our way to describe the boundedness theorem after firing one transition in the Mizar language is described as follows:

theorem :: Boundedness theorem for single transition
for Dftn being Decision_free_PT,
dct being Circuit_of_places_and_trans of Dftn,
M0 being marking of Dftn, t being Element of the carrier' of Dftn
holds num_marks(places_of dct, M0) = num_marks(places_of dct, Firing(t, M0))

The term "Decision_free_PT" in the above theorem stands for decision free Petri net structure whose Mizar definition is described as,

definition :: Decision free Petri net

```
 let IT be Petri_net;
 attr IT is decision_free_like means
 for s being place of IT holds ((ex t being transition of IT st [t, s] is Element of the
 T-S_Arcs of IT) & (for t1, t2 being transition of IT st [t1, s] is Element of the T-S_Arcs
 of IT & [t2, s] is Element of the T-S_Arcs of IT holds t1 = t2)) & ((ex t being transition
 of IT st [s, t] is Element of the S-T_Arcs of IT) & (for t1, t2 being transition of IT st [s,
 t1] is Element of the S-T_Arcs of IT & [s, t2] is Element of the S-T_Arcs of IT holds t1
 = t2));
end;
```

The above attribute called "decision_free_like" describes that the basic structure of Petri net is decision free if, for all element s $\in$ place (the set of all places of Petri net), $\exists t_1,t_2 \in$ transition (the set of all transitions of Petri net) such that the arc $[t_1, s] \in$ T-S_Arcs is the only input arc to s (i.e., for all $t_1, t_2 \in$ transition (the set of all transitions of Petri net) such that $[t_1, s] \in$ T-S_Arcs and $[t_2,s] \in$ T-S_Arcs implies t1 = t2) and the arc $[s, t_2] \in$ S-T_Arcs is the only output arc of s (i.e., for all $t_1,t_2 \in$ transition (the set of all transitions of Petri net) such that $[s, t_1] \in$ S-T_Arcs and $[s, t_2] \in$ S-T_Arcs implies t1 = t2). In short, each place in the net has exactly one input transition and one output transition. The term place, transition, S-T_Arcs, T-S_Arcs is borrowed from Mizar Library, whose description is presented in the section 4.1 of Appendix C.

Also num_marks in the above boundedness theorem is a function of places P and marking M that adds up the generated natural marking of the places in net, whose Mizar code is shown below.

```
definition :: summation of generated natural marking
 let N be PT_net_Str;
 let P be finite Subset of the carrier of N;
 let M0 be marking of N;
 func num_marks (P, M0) -> Element of NAT equals
 Sum (the Enumeration of M0, P);
end;
```

Here " the Enumeration of M0, P " is used to construct a finite sequence of the selected values of natural marking of the places in Petri nets, which in Mizar is described as,

```
definition
 let X be set;
 let Y be non empty set;
 let P be finite Subset of X;
 let M0 be Function of X,Y;
 mode Enumeration of M0, P -> FinSequence of Y means
 len it = len (the Enumeration of P) &
 for i st i in dom it holds
 it.i = M0.((the Enumeration of P).i) if P is non empty
 otherwise it = <*>Y;
 ...
end;
```

Here we used arbitrary X and Y instead of the carrier of Petri net and NAT, because this definition is useful not only in our work, but also can be useful another time in another formal verification work after it is stored in the Mizar library. With the carrier of Petri net and NAT, we would have restricted to only Petri net type and NAT. The notation Enumeration in the above definition means one-to-one finite sequence, no duplicates, and all elements of P is taken. Once we have a finite sequence of selected values of M0, we just apply functor "Sum" to this Enumeration to get the desired summation (num_marks) value. If P is empty, then the Sum is Zero from the definition of Sum. Here we added assumption that P must be finite, otherwise we can't construct the desired finite sequence.

Similarly places_of dct in boundedness theorem yields finite subset of the places in P, which in Mizar is defined as,

```
definition
 let PTN be Petri_net;
```

```
  let dct be Finsequence of places_and_transition_of PTN;
  func places_of dct -> finite Subset of the carrier of PTN equals
  {p where p is place of PTN : p in rng dct}
...end;
```

The Mizar code of the directed path and directed circuit is described below. In order to formalize directed circuit concepts, first we need to deduce directed path concepts.

```
definition :: directed path
  let PTN be Petri_net;
  let IT be FinSequence of places_and_trans_of PTN;
  attr IT is directed_path_like means
  len IT >= 3  & len IT mod 2 = 1 &
  (for i being Nat st i mod 2 = 1 & i + 1 < len IT holds
  [IT.i, IT.(i+1)] in (the S-T_Arcs of PTN) &
  [IT.(i+1), IT.(i+2)] in (the T-S_Arcs of PTN))
  & IT.len IT in (the carrier of PTN);
end;
```

The concept of a directed path is defined in Mizar as an attribute called "directed_path_like". For a finite sequence of places and transitions of a given Petri net, it is directed_path_like if there are at least three elements and the total number of elements in the sequence is odd. Also, the elements of the sequence will be ordered in such a way that every odd-numbered element (except for the last element) and the even-numbered element following it form a valid S-T_arc in the net, as shown in Fig. 3.1. Similarly, every even-numbered element and the odd-numbered element following it form a valid T-S_arc in the net, as shown in Fig. 3.2. Finally, the last element in the finite sequence of nodes (places and transitions) is a place element.

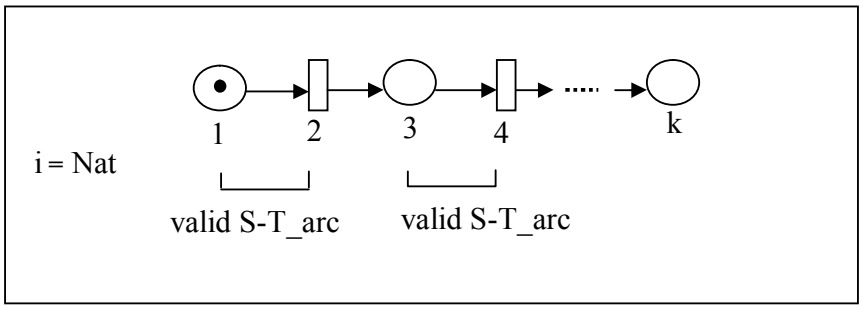

Figure 3.1. Valid S-T_Arcs in Sequence.

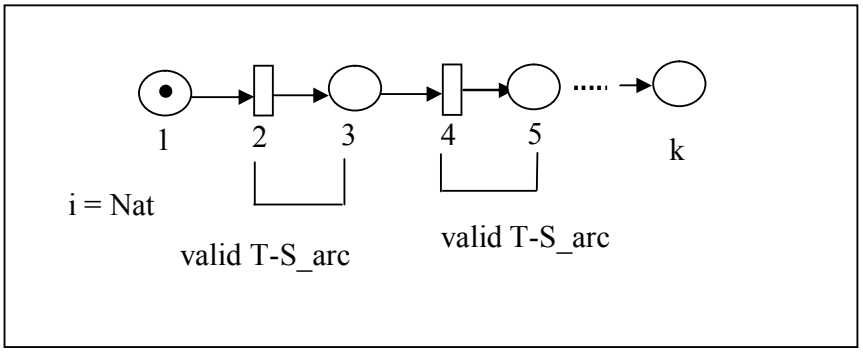

Figure 3.2. Valid T-S_Arcs in Sequence.

definition :: directed circuit
let PTN be Petri_net;
attr PTN is With_directed_circuit means
ex fs being FinSequence of places_and_trans_of PTN st
fs is directed_path_like &
fs is circular &
fs is almost-one-to-one;
end;

The term "Circuit_of_places_and_trans of Dftn" from the above token boundedness theorem represents that the directed circuit is a directed path and also circular and almost-one-to-one finite sequence of places and transitions of decision free Petri net. The definition of "circular" attribute is borrowed from article FINSEQ_6 in MML [19], whose definition states

that the first and the last element of the sequence is same (similar to the definition of a directed circuit of this work). Again, the definition of "almost-one-to-one" attribute is also reused from article JORDAN23 stored in MML [20], whose definition states that the sequence does not contain duplicate elements between the first and last elements.

3.2 Proof of the above token boundedness theorem

In natural language proofs, sometimes only the most convenient cases are addressed [1] as discussed in Chapter 2. But in Mizar, all possible cases must be explicitly proven before proving the particular theorem. In this case also when proving this theorem in Mizar, we realize that the available proof of this theorem in the book is only valid for the case of a circuit that has no repeated elements. So, it was necessary to prove the theorem for all cases by our own knowledge and experience. In the proof of the above theorem, we first checked for whether a circuit is empty or not, then we divided the cases of whether transition t ∈ T is firable at marking M0 or not, then whether transition t ∈ T is in the circuit or not. If transition t is in the circuit, then whether the length of the circuit is either equal to 3 or greater than 3. In all of the cases, we had to prove that the total number of marks in the places of the circuit is the same before and after the transition t firing operation. The basic outline of our proof strategy of this theorem is given in Fig. 3.3.

```
...
  if circuit is non empty
  ...{...
Case 1    t is firable at M0
          {
Case 1.1    t in circuit
            {...
Case 1.1.1   circuit length = 3
             {...
              num_marks(places_of dct, M0) = num_marks(places_of dct, Firing(t, M0))
              }
```

```
Case 1.1.2     circuit length > 3
                 {...
                 num_marks(places_of dct, M0) = num_marks(places_of dct, Firing(t,M0))
                 }
               }
Case 1.2       not t in circuit
               {...
                 num_marks(places_of dct, M0) = num_marks(places_of dct, Firing(t, M0))
               }
             }
Case 2         not t is firable at M0
               {...
                 num_marks(places_of dct, M0) = num_marks(places_of dct, Firing(t, M0))
                 }
               ...}
```

Figure 3.3. Outline of proof strategy.

This shows that if individual cases are omitted, the conclusion of the theorem can be false. The case in theorem is important because it makes the complicated proof easy and readable. This proof preserves the number of tokens in places of the circuit after firing a transition t. In order to prove this considered theorem, we need 15 theorems which is directly or indirectly connected to this theorem to be proved before proving a boundedness theorem, also 14 definitions and 1 lemma were necessary [54]. Lemmas are technical results used in the proofs of theorems. Often it is found that the same trick is used several times in one proof or in the proof of several theorems. When this happens the trick is isolated in a lemma so that its proof will not have to be repeated every time it is used. This often makes the proofs of theorems shorter and expressive. The description of each case with a diagram is shown below.

Case 1.1.1 len dct = 3

First of all, let us consider the case that the circuit is not empty, transition t is firable, transition t is in circuit, and the length of the circuit is equal to three.

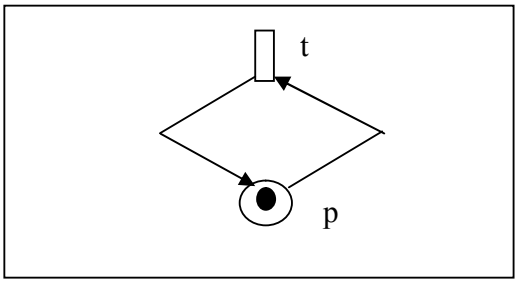

Figure 3.4. Directed circuit with length equal to 3

Here a place labeled by p is the only input place for transition t and the same place is also the only output place of transition t. Thus, according to transition enable and firing rule (refer Appendix B), this t is enabled and after firing t, token count does not change and thus preserves the number of token in a circuit of decision free Petri net structure.

Case 1.1.2 len dct > 3

Now, let us consider the case of non empty circuit where transition t is firable and available in circuit, and also the directed circuit length is greater than three. In Fig. 3.5, the finite alternating sequence of places and transition forms circuit, where p1 = pn.

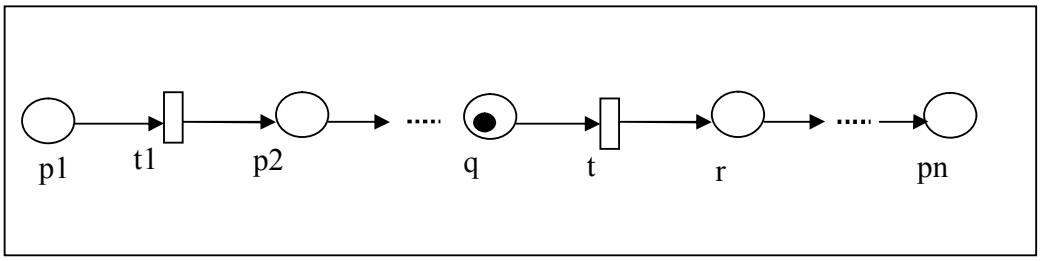

Figure 3.5. Directed circuit with length is greater than 3

In order to understand clearly, the element of Fig 3.5 in text can be written as
let dct = p1, t1, p2, ..., q, t, r, ..., pn . where q, r are places, t is transition.

Then in the Mizar article of this work, it has been proved that for every place s, if s is a place in dct and s is not equal to q then

1. not s in *'{t} (This means any place s is not the input place for t)

Similarly, we proved the fact that for every place s, if s is a place in dct and s is not equal to r then,

2. not s in {t}*' (This means any place s is not the output place of t)

We also proved the facts that

3. not q in {t}*' (This means q is not the output place of t)

4. not r in *'{t} (This means r is not the input place for t)

Thus we have facts that place q is the only input place for t and another place r is the only output place of t and denoted by:

5. q in *'{t} and r in {t}*' eq.1

Now, let us consider the variables mM0, mFM0, and p1, which store the following elements:

mM0 - marking of places in dct before firing t

mFM0 - marking of places in dct after firing t

pl - sequence of places of dct,

The order of places in pl can be different from the order of places in dct, so after finding the exact location of place q (denoted by nq) and r (denoted by nr) in the enumeration of p1, we considered two subcases:

1.1.2.1 nq > nr , and

1.1.2.2 nr > nq (analogue of 1.1. 2.1)

First, let us consider case 1.1.2.1, where the exact location of q in p1 is greater than the exact location of r in p1,

For example, let say we have

the Enumeration of places of dct = <* p1, p2,...,r, q, ..., pn*>

Then the marking of places in above sequence is written as

the Enumeration of M0, places_of dct = <* M(p1), M(p2), ..., M(r), M(q), ..., M(pn)*>

Then in order to preserve the number of tokens in a circuit, our Mizar code for this case is written in such a way that the above marking sequence will be divided into the way as shown in Fig 3.6 and 3.7. The Fig 3.6, first restricts the marking sequence to the first (nq - 1) elements.

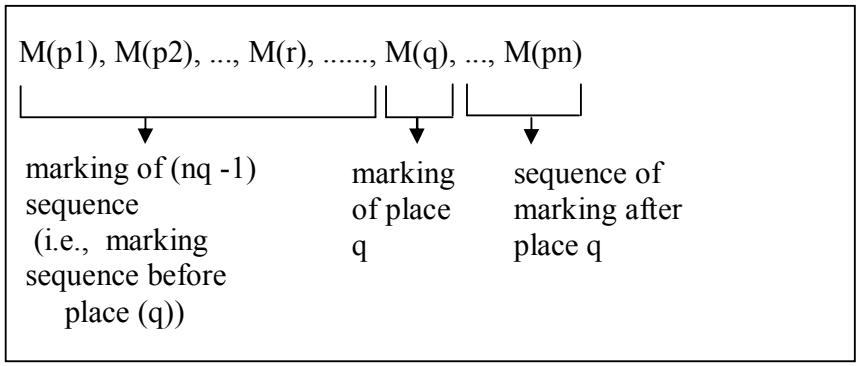

Figure 3.6. Restricting the marking sequence to (nq-1) elements

Then again the resulted marking sequence after restriction, i.e, the sequence of marking before place q, which is (nq - 1) in above Fig 3.6 is again divided into (nr - 1) marking sequence as shown in Fig 3.7.

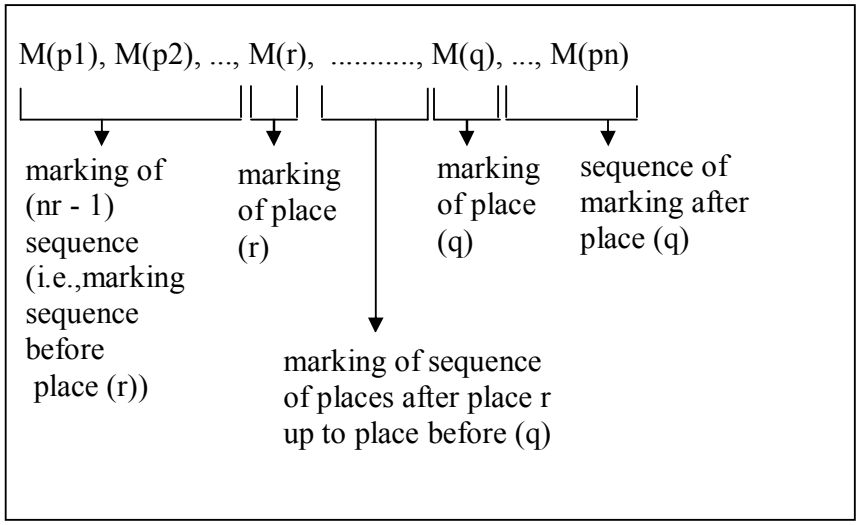

Figure 3.7. Restricting the resulting marking sequence (nq-1) to (nr - 1) elements

The Mizar code of the above process shown in Fig 3.6 and 3.7 is written in Fig 3.8. The total number of tokens in places of a circuit when nq > nr is calculated by following Mizar logical statements,

```
...
   num_marks ((places_of dct), M0)
   = (((Sum (( the Enumeration of M0, places_of dct | (nq - 1)) | (nr - 1)))
 +      (Sum <* (( the Enumeration of M0, places_of dct | (nq -1)) . nr)*>))
 +    (Sum (( the Enumeration of M0, places_of dct | (nq - 1))  /^  nr)))
 +    (Sum <*( the Enumeration of M0, places_of dct /. nq)*>))
 +    (Sum ( the Enumeration of M0, places_of dct /^ nq))
     ...
     (((Sum (( the Enumeration of M0, places_of dct | (nq - 1)) | (nr - 1)))
 +      (( the Enumeration of M0, places_of dct | (nq -1)) . nr) + 1
 +    (Sum (( the Enumeration of M0, places_of dct | (nq - 1))  /^  nr)))
 +      ( the Enumeration of M0, places_of dct . nq) - 1
 +    (Sum ( the Enumeration of M0, places_of dct /^ nq))
     ...
  = ((((Sum (( the Enumeration of Firing (t, M0), places_of dct | (nq - 1)) | (nr -' 1)))
 +       (Sum <*(( the Enumeration of Firing (t, M0), places_of dct | (nq - 1)) . nr)*>))
 +        (Sum (( theEnumeration of Firing (t, M0), places_of dct | (nq - 1)) /^  nr)))
 +      (Sum <*( the Enumeration of Firing (t, M0), places_of dct /. nq)*>))
 +       (Sum ( the Enumeration of Firing (t, M0),places_of dct /^ nq))
     ...
 = num_marks ((places_of dct),( Firing (t, M0)))
  ...
```

Figure 3.8. Mizar code for preserving the number of tokens when nq > nr

Here " the Enumeration of M0, places_of dct | (nq - 1)) | (nr - 1)" from Fig. 3.8 (Line 2) first restricts the marking sequence to the first (nq - 1) elements as shown in Fig. 3.6, and after

that it restricts the resulting sequence to the first (nr - 1) elements as shown in Fig. 3.7. " ((the Enumeration of M0, places_of dct | (nq -1)) . nr) " from Fig 3.8 (Line 3) gives the exact marking of place r (i.e, M(r)) from the resulting sequence (nq - 1). Similarly "((the Enumeration of M0, places_of dct | (nq - 1)) /^ nr)" expression from Fig. 3.8 (Line 4) divides the rest of marking sequence between M(r) and M(q). Again "(the Enumeration of M0, places_of dct /. nq)" expression from Fig 3.8 (Line 5) gives the exact marking of place q (i.e, M(q)). Then the expression "(the Enumeration of M0, places_of dct /^ nq)" from Fig 3.8 (Line 6) gives the rest of marking sequence after the nq-th element as shown in Fig 3.6 and 3.7. After dividing the marking, since according to eq.1 place q is the only input place for transition t and place r is the only output place of transition t. So, according to transition firable and firing rule, marking from place q will lose 1 token (i.e., M(q) - 1) and 1 token will be added to the only output place r (i.e., M(r) + 1), and all other marking sequences will be same as shown in Fig 3.6. Then, after summing all resulted new marking sequences, the total number of marking will again be same before and after firing the transition t.

Similarly, the whole process will be opposite in the case nr > nq, which also preserves the number of token before and after firing transition t. The detail Mizar code of this case is available online [17].

Case 1.2 transition t not in circuit.

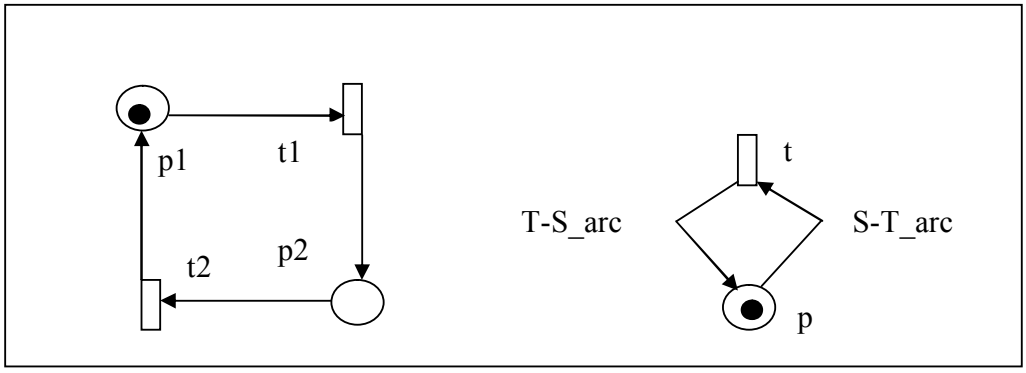

Figure 3.9: Any transition outside of Circuit

The transition t, which is not in the circuit will not affect the number of tokens in the circuit. The reason is because every place in the decision free Petri nets, there is exactly one incoming and one outgoing arc. Since every place in the circuit is connected with the transition of the same circuit. Thus, any transition t which is available outside the circuit means there is no any arc connection between any place inside the circuit and this transition t. And the relation of outside transition t with any outside place p forms, either S-T_Arcs or T-S_Arcs as shown in above Fig. 3.9.

Case 2. t is not firable

According to transition firing rule, if input place of any transition t does not hold any token means t is not firable. If t is not firable means there is no any token movement in the net.

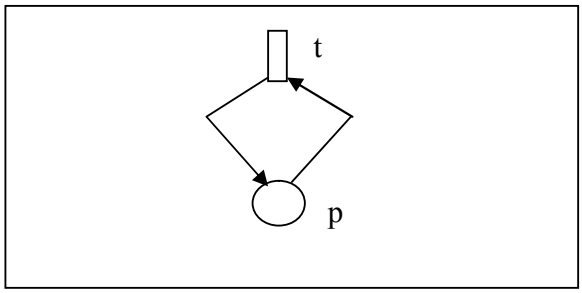

Figure 3.10. Transition t is not firable

Until now the above boundedness theorem and its proof of different cases is for firing single transition t. Finally, by using all of these definitions and several accompanying lemmas and propositions, we successfully proved token boundedness property for firing transition sequence by using induction statement. The Mizar code of the token boundedness property for transition sequence is given below.

theorem
for Dftn being Decision_free_PT, dct being Circuit_of_places_and_trans of Dftn, M0 being marking of Dftn, Q being FinSequence of the carrier' of Dftn holds num_marks (places_of dct, M0) = num_marks (places_of dct, Firing(Q, M0));

Chapter 4 Conclusion and Future Work

When definitions and theorems of mathematical knowledge are described in natural language, some important details are sometimes omitted from the descriptions leaving the reader to fill the gaps in order to understand. Moreover, it happens that the construct of a theorem proof in the modeling or analysis tool sometimes is not true for all possible cases because of ambiguity or the missed information.

In this dissertation, we used a mechanical proof checking system called Mizar, where Mizar language is used for writing mathematical definitions and theorems of a subclass of Petri net and Mizar verifier checked the logical correctness of a system. Here, by using some basic information related to Petri net available in Mizar library, we built the structure of a subclass of Petri net called decision free Petri net with the aim to prove the token boundedness property in a circuit of this structure for both firing transition and transition sequence. While in the process of formalizing, we found that there is some omitted case and information in the proof of natural language, which eventually was causing errors several times. In order to fix errors, we had to define and prove the missed facts in our Mizar work before proving the token boundedness property, because the theorem checking system does not allow incomplete proofs to pass.

Formal verification of mathematical concept is important for modeling and analysis of hardware and software systems and also for building a reliable repository of mathematical information that can be used as a foundation for developing more complex system theories. The advantage of system property analysis in a mechanical way is that each step in a formal language is explicitly given and justified, and hence it has a computer guarantee of logical correctness. Logical correctness plays a vital role in the system development process. Errors traced late in the development of computer hardware and software systems are not acceptable any more for both industries and designers because of high capital investment.

Future Work

In future, the definitions and theorems of decision free Petri net stored in the database of the theorem checking system can be used to analyze the liveness and safeness properties in directed circuits. Liveness property has a fact, that the directed circuits should cover all transitions, and the initial marking M0 assigns at least one token on each directed circuit.

Whereas, decision free Petri net is safe if and only if it is a live Petri net, the directed circuits cover every place, and also the initial marking M0 assigns exactly one token on each directed circuit.

Appendix A Formal Methods and its Approaches

In this section, we briefly introduce formal methods (FMs), its purpose, and its different approaches. It then familiarizes the reader with the tools and techniques used for formalization of mathematical knowledge and verifying its correctness. It then describes some of the theorem provers tools with its advantages and disadvantages.

1. Introduction

Mathematical language is a mixture of words and symbols that mathematicians use to write mathematical formula in textbooks mostly in natural language which can be implemented on the computer under the strict guidance of formal semantics. Over the past three to four decades, there have been a noticeable achievement in the area of theorem proof checking systems for reasoning mathematical definitions and theorem statements in tools. Formal proof systems have been well studied in industrial and academic fields to formalize standard or classical mathematical concepts. Because of its powerful and well organized implementations of logic, formal proofs are becoming more popular and relevant to many different fields that depends on mathematical reasoning techniques. No doubt, there is a visible difference between formal proof text and natural language texts. We can often see that the steps in the natural language description have a gap, which is definitely needed to be filled in order to be clearly understood for readers of all areas. This is because an expert in mathematics usually avoids writing such trivial facts that is obvious and repeated. They find it distracting and boring to give some small obvious details in each and every step. Whereas the reader of another area, not expert in mathematics, find it more sophisticated and complex since they believe that the correctness of mathematical texts should depend on the every statement stated before.

2. Motivation

We all know that there is a considerable amount of mathematical knowledge stored either in many different textbooks or in the brain of mathematicians. We always have difficulty to find all related mathematical knowledge and its reference in one place. One needs to follow different reference website or textbooks to understand sophisticated mathematical formula. This is the reason that we need an active system where we formalize theorems and definitions of well

formed algorithms and then all the information can be stored in one folder as an open digital library which can be referred by all users. The same information in the library can be browsed and searched by the user who wants to refer and extend the available knowledge to analyze their own interested model. Storing this knowledge in the right form of computers, the mathematics should be more readily available to be used either by programmers or by computer applications which is required as a pre definition or pre theorems for the justification of steps. The important advantage of creating the library is that all the information stored there has a computer guarantee of logical correctness of statements.

Before embarking on the detailed description of FMs, let us be familiar with the taxonomy (types and approaches) of it that can be used for modeling and checking the system as given in Figure 4.1.

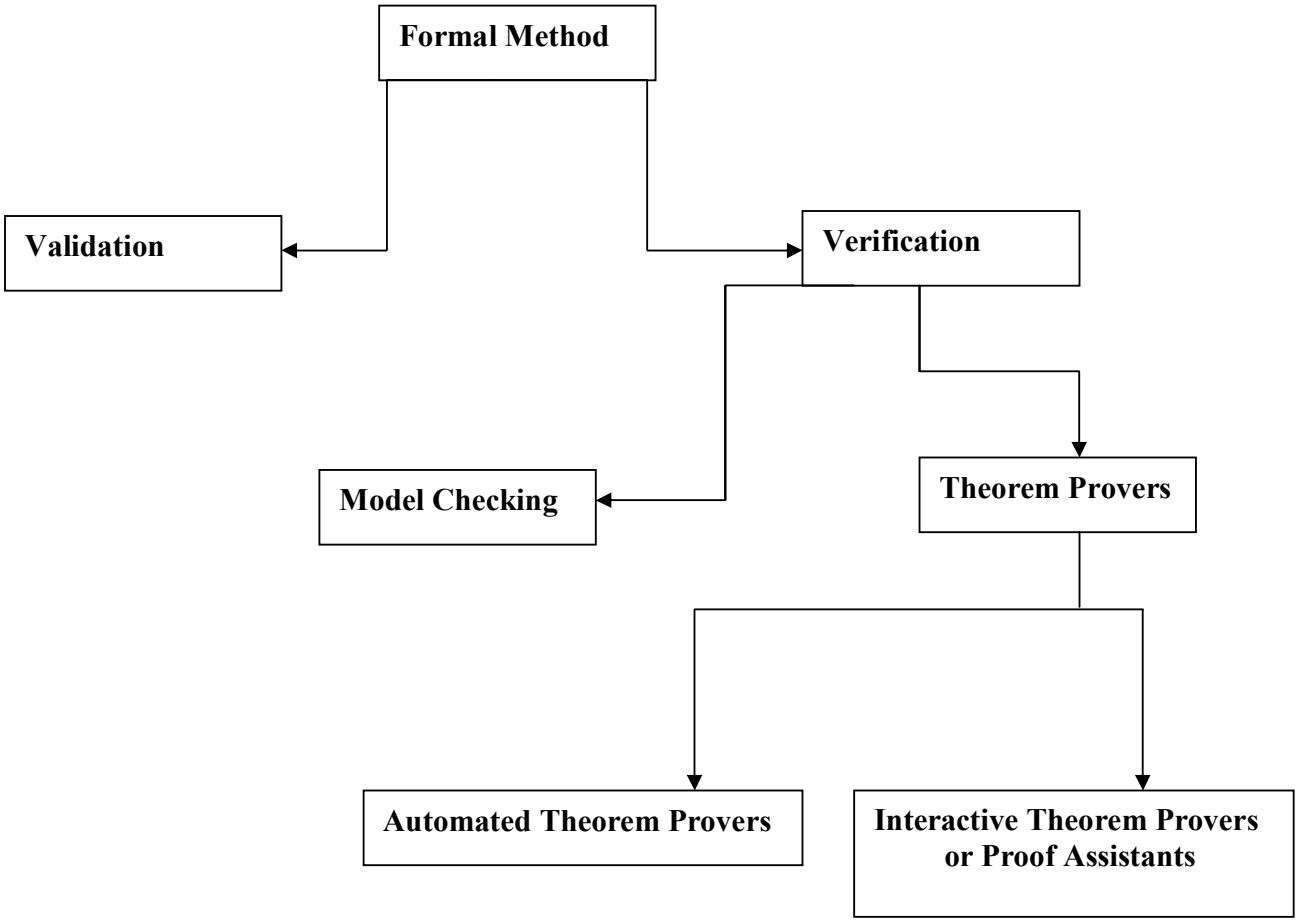

Figure 4.1. Formal method and its approaches

3. What is FMs?

FMs in system development engineering are a mathematical and logical based techniques which can be used for the specification, verification, and implementation of a complex hardware and software program [21], [22]. In the specification phase, designer rigorously defines a system using a modeling language. The process of formal specification is similar to the process of converting a word problem into an algebraic notation. Formal verification is a process where designers are heavily emphasized on analysis or proving the correctness of a property of systems on the specified model. Whereas implementation is a method of converting the specification into code. When models become more sophisticated and where safety is an important issue, the formal methods for system development assures another level of insurance. The proofs by this method can be performed automatically using automated or interactive (proof assistant) theorem provers (TPs) by different available tools. The reason for studying formal methods is the expectation to create reliable and robust hardware and software programs.

4. Why FMs?

For many years, researchers have been using FMs to generate a better software and hardware process and increase its quality. The advantage of this method includes finding system defects earlier, checking of structural and behavioral properties of systems, and decreasing the redesign work. This method can also help the researchers to avoid or reduce the ambiguity which is an intrinsic phenomenon of natural language, FMs is even more important in the fields of system where the failure of software cause loss of life and cost. One example of life loss among several is, the crash of Air France flight 447 in 2009, killed 228 people, was partly due to discrepancies in the indicated airspeed readings [23]. Similarly, another example of budget loss among several is, the presence of bugs in an automatic trading program cost its proprietor 440 million dollars in 30 minutes when it was deployed on August 1st 2012 [24]. The causes of these failures is because of a lack of guards against unexpected inputs and mismatch between specification and actual software, which could have been prevented by strict coding as well as thorough reviews by computer system.

The reason of formal method raised two fundamental questions. First one states that: Are we building the right model? In order to answer this question, the model needs to be validated [25]. The second question states that: Are we building the system right? The answer of this

question needs model verification. In other words, validation is concerned with checking that the system meets the customer's necessity, while verification is involved with whether the system is well-designed, and so on. Verification process helps designers to determine whether the software system is of high quality and bug free which is mainly required for building a complex system.

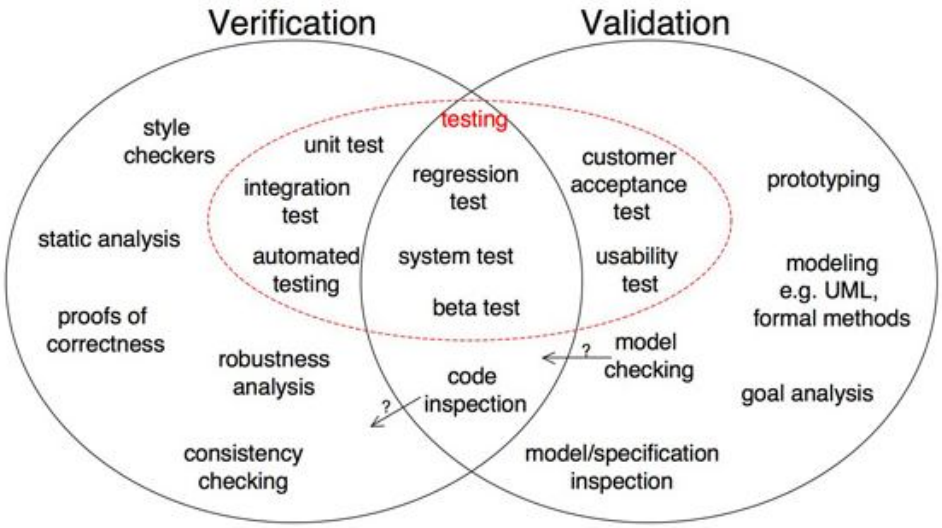

Figure 4.2. Verification and validation (Source: [46])

5. Formal verification

Formal verification is defined as an approach to create a mathematical model of a system using a formal language to specify desired properties of the systems in a concise and unambiguous way, and using a method of proving and verify that the specified properties are satisfied by the model. When the method of proof is carried out substantially by machine, the method is called automatic verification. In this thesis, the emphasis is done on the formal verification, and more specifically, on the formalization of mathematical definitions and theorems and proving its logical correctness. Formal verification of hardware systems has been common by using model checkers and TPs [27], [28]. Verifying the correctness of a hardware and software program requires formalization of a functional property to be verified using a suitable logical formula such as higher order logic, first order logic or temporal logic [29].

6. Model checking and Theorem Proving System

6.1. Model Checking

Model checking is an approach where a finite model of a system is built and checked against a set of desired properties. In another word, model checking is a technique where the specification is expressed as a finite state model [30]. Here the system is modeled as a set of states together with a set of transitions between states that describe how the system moves from one state to another in response to internal or external stimuli. In model checking approach, a system problem or the desired properties involve the construction of abstract model in the form of either temporal logic [31] or on finite state automation [32], as shown in Figure 4.3. Here an exhaustive search of the state space is performed in order to check that the system is a model of its specification (or to check weather model satisfies its specifications).

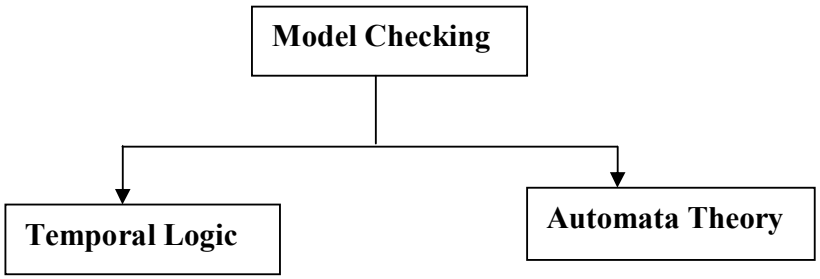

Figure 4.3: Model Checking Techniques

The advantage of the state based exploration technique is that once the correct design of the system and the required properties is built, the verification process is typically fast and fully automatic. The other advantage is that if the design or event of the property does not hold true in some states, the verification process generates a counterexample file with important debugging information. This approach is more limited in scope than the other automated formal techniques (example, deductive methods, like TPs), but is intuitive, fast, and fully automated, and thus is popular in many industrial areas. The specification of the system is very similar to programming, so it does not require much additional expertise from the user. The example of model specification and model checking tools include standard specification language such as LOTOS or SDL (supports CCS and CSP process) [33], CADP (specified in the ISO language LOTOS)

[33], [34], and there are many more to list. There are Communication Sequential Processes (CSP) as a model of massage passing parallel computation. The ISO standard LOTOS abbreviated for Language of Temporal Ordering Specification is a process algebra used to describe CSP and synchronizing by rendezvous on gates. LOTOS was developed as a language by adding abstract data type to CSP. It seems to be suitable for the connection of multiple arithmetic elements which perform the concurrent operation and the description of their behaviors.

Knowing the fact that this model depends on the search of the state space of the system, unfortunately this may increase with the size and complexity of system description, commonly known as the state space explosion problem. This is because of the exponential relation of the number of states in the model to the number of components, that results system states. This restricts the scope of model checking techniques for sophisticated systems.

6.2. Theorem Proving

Theorem proving is an approach where a system is modeled as a mathematical definition in some formal mathematical mechanical rules and logic. The desired properties of a system are then derived as a theorem, that needs to be proved, using the formalized definitions and lemmas. Here every line of the proof is explicitly justified, and hence is fully completed and readable. Theorem proving tools, mainly depend on higher order logic, first order logic, set theory, constructive logic, etc. It has two approaches, that is, automated theorem provers and interactive theorem provers. Before giving the detailed definition of its two different approaches, let us know about what is a proof and why is needed.

a) Proofs.

Webster's dictionary describes that "the proof is a process of establishing the validity of a statement, especially by derivation from other statements in accordance with principle of reasoning". A mathematical proof reduces to a series of very small and simple steps and all these steps can be justified. These steps are so small and obvious that no mathematicians want to write this in their textbooks, but it generally holds the position that mathematical proof that we find in books can have some incomplete details. It sometimes happens that a mathematical theorem from literature turns out to be false. In such case their formalized proofs and theorems cannot be verified, and also takes extremely longer duration to be verified. This is the reason that

mathematicians also agree on the validity of the basic proof steps. This achievement encourages researchers to use mechanical proof formal verification approach.

6.2.1. Automated theorem provers (ATPs)

This is a subsection of theorem provers. Automated theorem provers (ATPs) are programs that are built for automatically finding proofs of formulas or theorems, rather than checking the proofs formalized by humans. One can also define this approach as a system consisting of a set of well chosen decision procedures or rules that allow formulas of a specific restricted format to be proven automatically. One of the main drawbacks of this system is that the generated proof is large enough even for a simple system. Another disadvantage of ATPs is its inherent low/limited expressiveness of their input language.

6.2.2. Interactive Theorem Provers (ITPs)

In this subsection, we describe computer programs that have somewhat different nature than the formal mathematical systems described in the previous subsections of automated theorem proving systems. Interactive theorem provers (ITPs), also known as proof assistant tools, are programs that help us to define the mathematical definitions, theorems, and do logical step by step reasoning by hand on the computer. The main aim of this approach is to do proofs. This approach is used mainly by researchers or specialists who formalize mathematical definitions in it and prove theorems. Proof assistants differ from automated theorem proving systems in that proof assistants help developers to develop human readable proofs that have computer guarantee of logical correctness. Since these proofs are mechanically checked by both human and computer, they do not share the controversial implications of computer aided proofs by exhaustion.

6.2.3. Declarative and Procedural input Language

The input language of the proof assistants can be a declarative or procedural style. The structure of a declarative proof style is more natural, i.e. it has a closer structure to the language used to prove mathematical formulas in textbook mathematics. Declarative style proof is more readable than procedural style proof because declarative proof consists a list of steps, where each step in the chain of logical statement is fully justified from the previously justified statements in the proper order.

On the other hand, in the procedural system, the user issues commands that transform so-called proof state until it reaches a state where the proof is finished. In this process, the user mostly exhibits the transitions of this system rather than the intermediate states that it goes through.

As discussed above that the declarative style is more readable because the script explicitly contains the successive proof states, also more accessible because it relies on a few basic operations, easier to understand because this language is quite close to the natural mathematical language style, and much maintainable because modifications affect the behavior of the script only locally. Whereas proofs in procedural system is less readable for the human user because proofs are scripts of commands. In this thesis deductive proof assistant approach is chosen for formalizing the case study.

6.2.4. Some Available Proof Assistants

The three main classes of proof assistants are:

a) That based on Church's higher order logic. Example of proof assistants falls into this category are: HOL, Isabelle/HOL, ProofPower, and maybe also systems like PVS,

b) That based on Martin-Lof's type theory. Some example of this proof assistants are Coq, NuPRL, Agda, Epigram.

c) and that based on set theory. The proof assistants belongs to this category are: Mizar, Metamath, Isabelle/ZF.

Below is the brief description of some among several available and active proof assistants tools used for formal verification:

A). HOL

HOL is an interactive theorem prover for checking the correctness of mathematical proofs of systems program, developed by Mike Gordon in 1980s [35]. This system is based on higher order logic: a programming circumstances in which mathematical theorems can be proved. Here the formalization process is no longer completely manual, but supported by a computer system. HOL Light is a latest version of the HOL theorem prover. This is famous for Kepler's conjecture proof [36]. HOL Light is coded in CAML Light with the ability that it can run well even on small machines, e.g. PCs and Macintoshes with only a limited megabyte of RAM. This

system is widely used for proving the correctness of mathematical model. The system is adapted to the expertise and needs of computer scientists, and does not have intention to resemble the reasoning and language of formal mathematics closely.

B). ISABELLE

Isabelle is also an interactive proof assistant tool, developed by Larry Paulson, Tobias Nipkow and Makarius Wenzel as a successor of HOL, based on LCF style which is written in standard ML. This system is also developed and used for checking the correctness of mathematical proofs of systems. The important difference between HOL and ISABELLE is that the latter one did not hardwire the mathematical foundations into the system, but keeps it as a parameter of the system, whereas the HOL implementation is on top of the Isabelle, called Isabelle/HOL. One more difference between HOL and Isabelle is that Isabelle has a readable proof language inspired by the Mizar language called Isar.

C). MIZAR

Mizar is a system initiated by Andrzej Trybulec in 1973 [37] at the university of Bialystok. The case study of this work presented in this dissertation is totally based on this system. The detail description of the Mizar system and its input language is discussed in the Appendix C.

Appendix B Petri Nets

Here, after describing the motivation of formal verification of Discrete Event Dynamic Systems, some preliminary notations and symbols of the basic Petri nets will be introduced. After that a subclass of Petri nets, i.e., decision free Petri net will be discussed. This chapter also gives some description of its substructure called directed path and directed circuit.

1. Motivation

Because of the high demand for increased productivity, flexibility, and competitiveness, the world of modern society has created a high performance discrete event dynamic systems (DEDS's). Concurrency and synchronization are important and well studied behaviors in DEDS's. These systems are often complex and larger in scale, and thus highly capital invested systems. A lot of work goes into modeling, analyzing, and verifying the structure and behavior of these systems. Logical correctness plays a vital role in the design and operation of such high performance DEDS's [2].

2. Petri Nets (PTNs)

PTNs are widely accepted and useful mathematical models for designing and analyzing a DEDS [1], [15]. Because of the concurrent and synchronization properties of Petri nets and their visually graphical notation, they hold the popularity for modeling DEDS's. The Petri net system may be computer hardware and software or a combination of both. Originally, PTNs were created by Carl A. Petri in 1962 for the study of communication with automata [38]. Further, several extensions of it and the restrictions on it has been studied for concurrency, synchronization, etc. [15]. Because of Petri nets graphical and mathematical representation nature, it has been popular and successful in the area of computer hardware and software system development.

The graphical representation of PTNs is easy to understand by readers, on the other hand, the ambiguous textual representation or mathematical notation of PTNs is difficult to understand. But as the complexity of the graphical representation increases and suffered by state space explosion problem, one has to move for formal analysis to check the correctness and reliability of the system. A mathematical Petri net model is described by a set of linear algebraic equations.

The formal analysis of Petri nets makes it possible to describe a strong statement of the properties of the process being modeled. This in turn prevents the existence of ambiguity, uncertainties. Among several behavioral properties of Petri net, token boundedness property is one of the well studied property of the system. The formal analysis of a model is especially required in the area of critical systems where safety is one of their primary requirements [39], [40]. In the area of manufacturing systems, PTNs have been used to model and analyze and to represent a simple production line with buffers which can be used for the study of boundedness and liveness property, performance evaluation for throughput, delays, capacity, etc [41].

3. Basic Concepts of Petri Nets

3.1 Formal notation of Petri Nets

This section provides the preliminary notations and symbols to be used in this work for those readers who are not familiar with PTNs. A place-transition net (PT-net) is a directed graph consisting of two sorts of nodes called places and transitions, such that no arcs connect two nodes of the same type. Graphically, a place is denoted by a circle, a transition by a box, and an arc by a directed line as shown in Fig. 6.1.

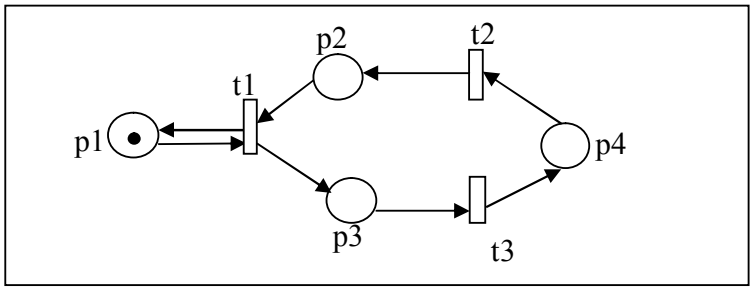

Figure 6.1. Petri net structure.

A Petri net is usually used to represent a discrete event system, where the places denote conditions, the transitions denote events and the arcs between places and transitions denote the relationship between conditions and events. Mathematical definitions of the basic structure of PTNs and its elements are:

Definition 1. The basic structure of a PT-net is described by a 3-tuple $N = \langle P, T, F \rangle$,

where, P is a set of places,

T is a set of transitions, and

$F \subseteq (P \times T) \cup (T \times P)$ is a flow relation from places to transitions and vice versa.

3.2. Pre-set and Post-set in Petri net

The set of places and the set of transitions in Petri net structure are disjoint, denoted by ($P \cap T = \emptyset$), means that there are no any common elements. Pre-set and post-set is the condition that is required during system model. The set of all input places of any transitions, denoted by •t symbol, is called the pre-set of transition and the set of all output places from any transition t is called post-set of transition t and denoted by t• symbol. Again, the set of all input transitions is called pre-set of any places and denoted by •p, similarly the set of all output transitions from any places is called post-set for any places and denoted by p•. Mathematically, it can be represented as:

Definition 2. A Petri net N is a structure, where its preset and post set element is denoted by,

$N = \langle P, T, F \rangle$ be a PT-net.

$\forall x \in (P \cup T)$,

${}^\bullet x = \{ y \mid (y, x) \in F \}$, is called the pre-set of x, and

$x^\bullet = \{ y \mid (x, y) \in F \}$ is called the post-set of x.

For clarity in presentation, the pre-set and post-set of a set of places or transitions, $X = \{ x_1, x_2, ..., x_n \} \subseteq (P \cup T)$, can be written as ${}^\bullet X$ and $X^\bullet$ respectively, where ${}^\bullet X = {}^\bullet x_1 \cup {}^\bullet x_2 \cup ... \cup {}^\bullet x_n$ and $X^\bullet = x_1^\bullet \cup x_2^\bullet \cup ... \cup x_n^\bullet$.

4. Dynamic Model of System

The description until now is a static structure of a Petri net. When the Petri net model changes, its state from one state to another, we called it exhibits the dynamic behavior of the system. In practice this is achieved through the use of marking. Marking of a Petri net means assigning tokens to the places on the net. A Petri net is a PT-net where tokens are assigned to its

places by a token distribution process called initial marking of PTNs. Mathematically, it can be represented as,

Definition 3. For a PT-net N = ⟨ P, T, F ⟩, a marking is a function M: P → {0, 1, 2, ...}, where M(p) is the natural number of tokens in p ∈ P. (N, M0) represents a PT-net N with an initial marking M0.

M can also be viewed as a vector given by $M_k = \{M_1, M_2, ...M_i, ..., M_n\}$, where the i th entry of M is M_i, which is the marking of the place p_i.

Example

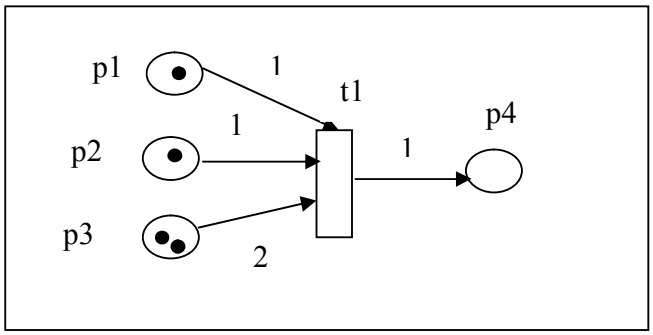

Figure 6.2. Tokens distribution in places

Places of Fig. 6.2, P = {p1, p2, p3, p4}

The initial marking at different places of the net is represented by

M:P(p) : {1, 1, 2, 0}

4.1. Transition enabled condition

Semantically, a marking represents the dynamic state of a Petri net. The initial marking specifically represents the initial state of a Petri net. State change in the Petri net model depends on the number of marking associated with each place and by the firing rules of transition. A transition t is said to be enabled if and only if all of its input places contains at least as many tokens in it as the weight of the input arc. Mathematically, an enabled transition is represented by,

Definition 4. For a PT-net (N, M0), a transition t is said to be enabled at a marking M0 if and only if ∀ p ∈ $^{\bullet}$t : M0(p) ≥ W(p,t),

where, W represents the weight of an arc from places to transitions. Transition t1 of the above Petri net diagram in Fig 6.2 is enabled because every input places of t1 contains the equal number of tokens as the arc weight to the transition.

4.2. Transition Firing Rule

An enabled transition has a potential to fire. Firing enabled transition removes tokens from the set of its input places and adds tokens to the set of its output places, which as a result gives a new marking from M(p) to M′(p). The dynamic behavior of the system is shown by the change in marking. Since the enabled transition can only fire, the number of tokens at any places is always non-negative. Generally, the number of tokens in net after firing might not be same as the number of tokens before transition firing. Firing of transitions can only take place as long as there is enabled transition in the net and if there is no enabled transition the execution stops. More than one transitions in a net can be enabled at a single time, but only one among them can fire. Also firing one transition may disable another transition which was firable before.

Example: The changes of marking in places after firing transition t1 is shown below,

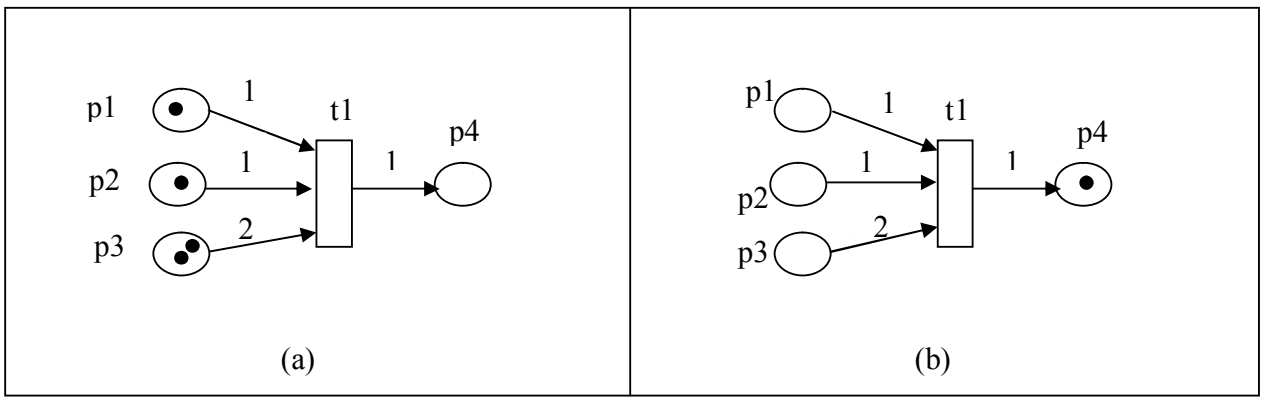

Figure 6.3. Before and after firing Transition t1

Mathematically, it can be defined as:

Definition 5. For a PT-net (N, M0), firing a transition 't' changes a marking M0 to new marking M'. i.e., $\forall\, p \in P$:

$$M'(p) = \begin{cases} M0\,(p) - W\,(p,t), & \text{if } p \in {}^{\bullet}t \ \&\ \text{not } p \in t^{\bullet}, \\ M0\,(p) + W(p,t), & \text{if } p \in t^{\bullet} \ \&\ \text{not } p \in {}^{\bullet}t, \\ M0\,(p), & \text{otherwise.} \end{cases}$$

Notice that a place in ${}^{\bullet}t \cap t^{\bullet}$ is marked whenever t is enabled, but does not change its token count by firing of t.

4.3. Transition Sequence

In order to find out whether the modeled system can verify the certain behavioral properties, it is necessary to find such a sequence of transitions firing, which would transform a marking M0 to Mn, where Mn represents the specific state, and the sequence of firing represents the required functional behavior. A marking M1 is said to be immediately reachable from M0 if firing an enabled transition in M0 results in M1.

Definition 6. For a PT-net (N, M0), firing of a sequence of transitions $\sigma = \langle t1, t2, t3, ..., tn \rangle$ transforms an initial marking M0 to Mn as shown below in Fig. 6.4.

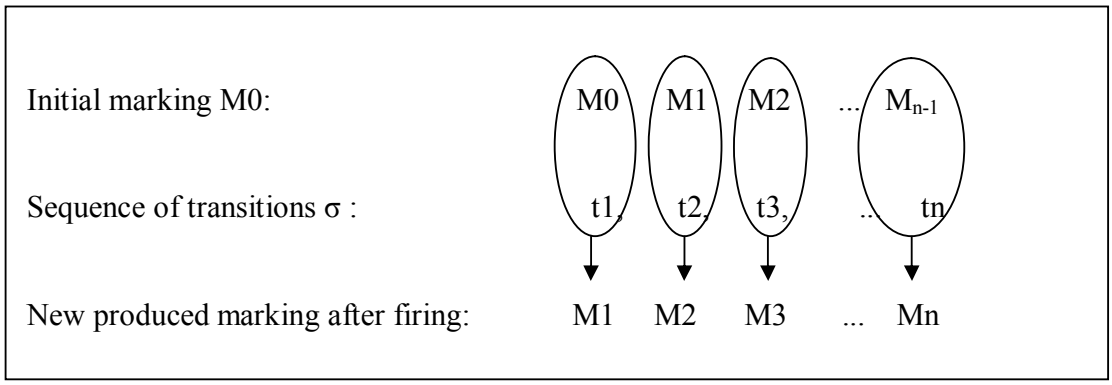

Figure 6.4. Firing of sequence of transitions

Here, firing of the first element of σ, i.e., t1, at initial marking M0 produces a new marking M1. Then, firing the second element of σ using M1 will produce a new marking M2 and so on until the firing of the last element of σ is done using Mn-1 to produce Mn which is the result of firing σ on Petri net starting from M0.

5. Representational Power in Petri Nets

This section describes some basic situations that can be required during modeling of a real Petri net model. Below are the Petri net structures that are used for representing characteristics of DEDS activities.

5.1. Sequential Execution

Sequential execution imposes the precedence constraint among the transitions in net. In the Fig 6.5, transition t2 can fire only after the transition t1 fires.

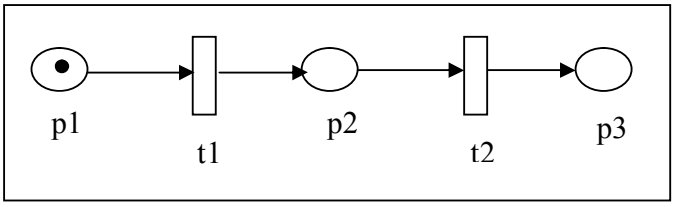

Figure 6.5. Sequential

5.2. Synchronization

Sometimes some parts of the system are waiting for resources to arrive. Once the resources are in input places the activities synchronizes all the resources as shown in Figure 6.6. Here t1 is enabled only if both p1 and p2 receives a resource, that is token.

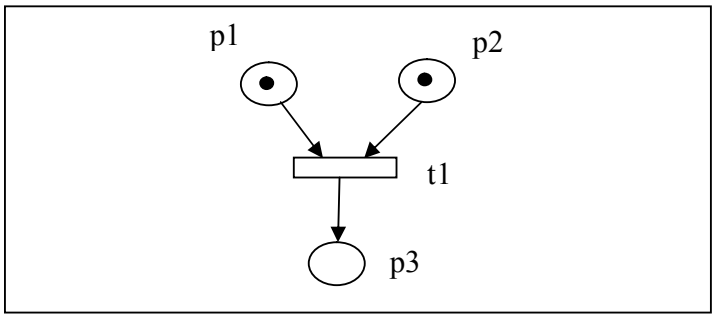

Figure 6.6. Synchronization

5.3 Concurrency

Concurrency is an important approach while building a complex system. Transitions t1 and t2 are concurrent in Figure 6.7. A transition is said to be concurrent if and only if there is a presence of fork style transition structure in the net which after firing produces a token in two or more output places in the net.

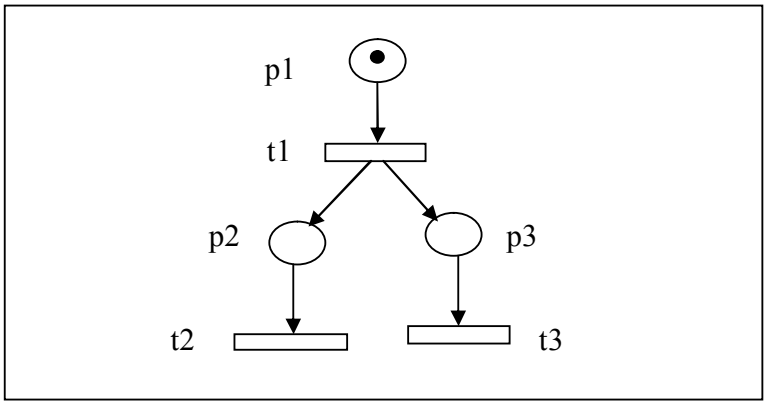

Figure 6.7. Concurrency

5.4. Merging: When parts from different branches arrive for processing at the same machine results merging. This diagram shows the merging situation.

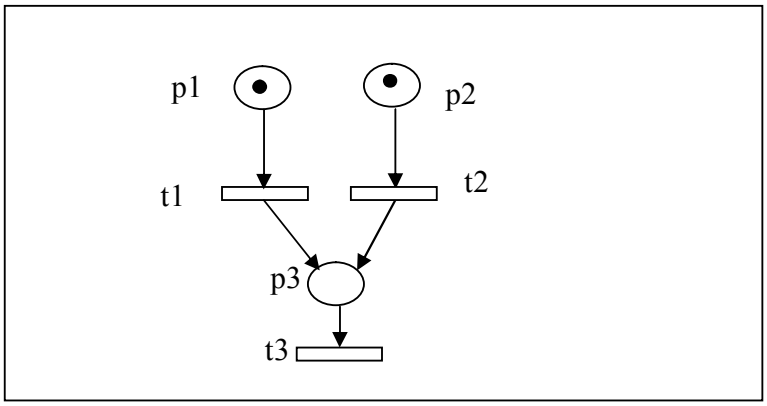

Figure 6.8. Merging

6. Substructures of Petri Nets

Because the behavioral properties of a system depend on its initial state and the

subsequent resulting states, Petri net models often become unmanageable even for a modest complex system and thus analyzing boundedness properties in such types of systems can become a daunting task. Hence it may be helpful if one can identify some substructures in the entire large Petri net and then it would be comparatively simple to model and analyze those substructures. Directed paths (DPs) and directed circuits (DCs) are two among several substructures of Petri nets used for these purposes. A directed path (DP) in a Petri net is defined as a finite alternating sequence of places and transitions present in the net. Similarly a finite sequence of places and transitions in the net is defined as a directed circuit only if the sequence is a directed path and the first and last elements of the sequence are same. The mathematical definitions of DPs and DCs in a net are given as:

Definition 7: For a PT-net (N, M0), a sequence of places and transitions, $p_1t_1p_2t_2...p_k$, is a directed path if and only if transition t_i is both an output transition of place p_i and an input transition of place p_{i+1} for $1 \leq i \leq K-1$. Pictorially, it can be represented as,

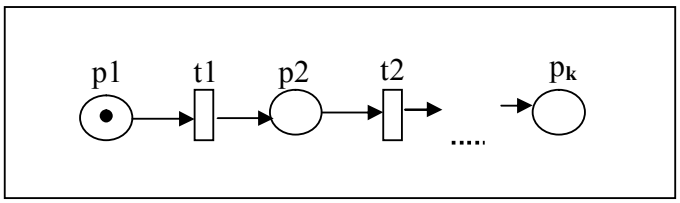

Figure 6.9. A directed path in a Petri Net.

Definition 8: In a Petri net, a sequence of places and transitions, $p_1t_1p_2t_2...p_k$, is a directed circuit if the sequence $p_1t_1p_2t_2...p_k$ is a directed path and p_1 is equal to p_k.

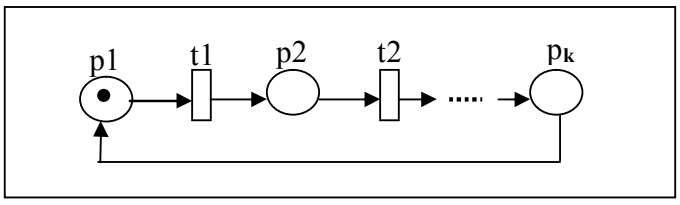

Figure 6.10. A directed circuit in a Petri Net.

7. Subclasses of Petri nets

There are a number of interesting subclasses of Petri nets. These subclasses play an important role in the development of certain application of Petri nets [44], [45]. Proving the boundedness property in the whole Petri net model is, in general, exponential-time, because of its complexity [41]. However, specific subclasses of nets like decision free Petri net and state machines restrict the net structure in such a way that certain properties can be proved using efficient algorithms [1], [42]. Boundedness properties are always necessary to be able to keep the Petri net model of the manufacturing system bounded, which is often needed for a well designed system because in the absence of this property, goods could accumulate without limit, which is often a design error [10].

7.1. Decision Free Petri nets (DFPNs)

DFPNs (also referred as marked graphs or event graphs) is the structure used in this dissertation to verify the token boundedness property using the Mizar proof checker. DFPNs are one among several subclasses of Petri nets that can model decision-free concurrent systems [1]. A Petri net is said to be a DFPNs if and only if for each place 'p' in the net, there is exactly one input and exactly one output transition with unique weight [1]. A transition may have multiple input places and output places. In this sense, decision free Petri net allows concurrent and synchronization structure. A DFPNs allows no conflict. A graphical representation of a decision free net is shown in Fig. 6.11.

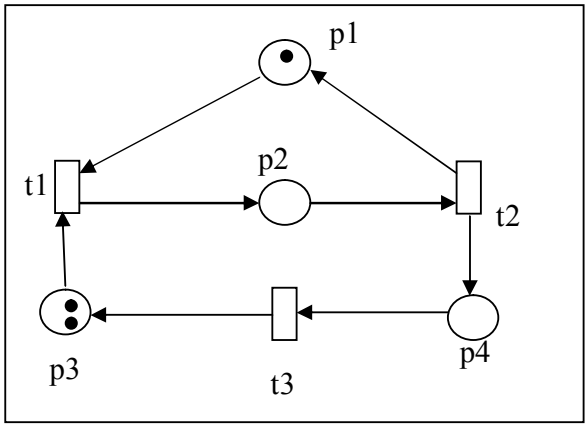

Figure 6.11. A decision free Petri net.

Mathematically, this net can be defined as,

Definition 9. PT-net N = (P, T, F) is defined as a decision free Petri net if and only if ∀ p ∈ P: |•p| = |p•| = 1, where, •p ∈ P represents the set of input transition t ∈ T to p, and p• ∈ P represents the set of output transition t ∈ T of p. Here 1 represents that each place in decision free net, there is only one input transition and only one output transition.

8. Properties of Petri Net

The mathematical model of Petri net possesses a number of properties. Among several Petri net properties, token boundedness property is one.

8.1. Token boundedness. It is defined as the preservation of the number of tokens in the places of the circuit of the net. The theorem with its proof and figure from literature is shown below.

Theorem: For a decision-free Petri Net, the number of tokens in a circuit remains the same after any firing sequence.

Proof: Tokens in the circuit can only be produced or consumed by transitions in the circuit. When a transition consumes a token, it produces one back into the circuit; therefore, the number of tokens in a circuit remains the same after any firing sequence.

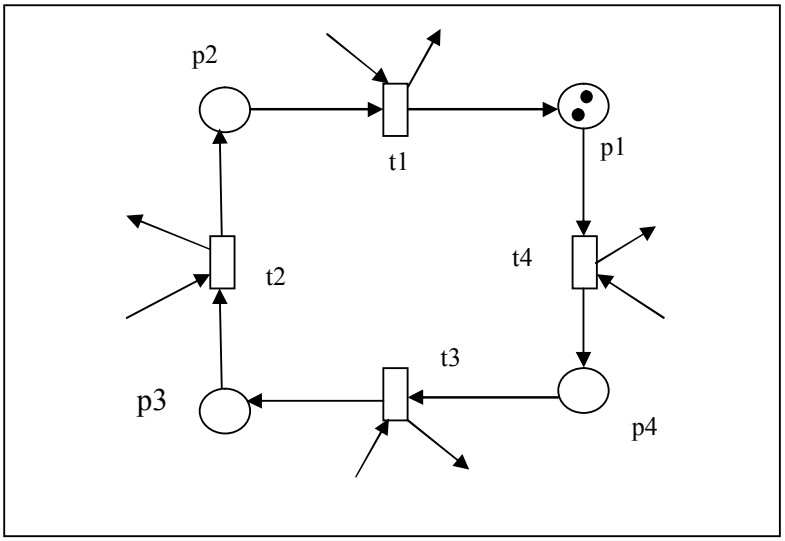

Figure 6.12. A circuit contains four transitions and four places

Appendix C MIZAR System and its Language

This appendix section contains background information on Mizar. It will also discuss about some Petri net knowledge that is already formalized in Mizar and available in its digital library.

1. Introduction

Mizar is one developed by Andrzej Trybulec since 1973 at the university of Bialystok in Poland [42]. The Mizar system consists of the Mizar language for writing mathematical definitions of formulas in a declarative style being close to the ordinary/informal mathematical texts and a proof checker software program to check files written in the Mizar language for logical correctness. Mizar project has been enough advanced because it has the world's largest repository of digital library of verified mathematical knowledge called the Mizar Mathematical Library, MML. This Mizar digital library contains material from various mathematical areas based on a single system of axioms. This system is based on Tarski-Grothendieck set theory with classical logic [43].

2. Formula and Skeleton of Proofs in Mizar

Mizar is essentially based on first order predicate logic, so the mathematical statements in Mizar are composed of predicate formulas combined with classical logic connectives and quantifiers. First-order predicate logic allows quantifiers to range over objects (terms), but not properties, relations, or functions applied to those objects. On the other hand, some theorem provers based on second-order predicate logic allows quantifiers to range over predicates and functions as well. The disadvantage of second-order predicate logic is that, there is no complete deduction system, also here reasoning is more difficult than in first-order predicate logic, because second-order predicate logic needs ingenious reference rules and heuristics. Inconsistency can arise in a higher-order system if semantics are not clearly defined. Where as in deduction style first-order predicate logic is well organized and easy to understand because it is very close to textual mathematical language that is used for describing mathematical definitions and proofs in literatures. The table below shows the Mizar representation of standard logical connectives and quantifiers:

¬ α	not α
α ∧ β	α & β
α ∨ β	α or β
α →β	α implies β
α ↔β	α iff β
∃xα	ex x st α
∀xα	for x holds α
∀x:αβ	for x st α holdsβ

Table 1 Standard logical connectives and quantifiers

3. Demonstration of very simple Mizar code

This section shows how Mizar formalizes mathematical formula. Let us consider we have formula like, " X ⊆ Y & Y ⊆ Z ⇒ X ⊆ Z ", which will be written in Mizar as

X c= Y & Y c= Z implies X c= Z;

Proving this formula in Mizar system is categorized into two blocks: environmental declaration block started by "environ" keyword which consists some directives (like vocabularies, notations, constructors, requirements, schemes, definitions, theorems, etc.) composed of a list of articles whose job is to unfold notations, definitions, and theorems to the present writing article and the next block is a text declaration block started with "begin" keyword. Everything that we need for our recent article like predefined symbol, symbol format, definitions, and theorems as named above, we need to add an appropriate article reference (already deposited in the MML) to each and every section of directives. The text block contains definitions and formula that a user wanted to prove.

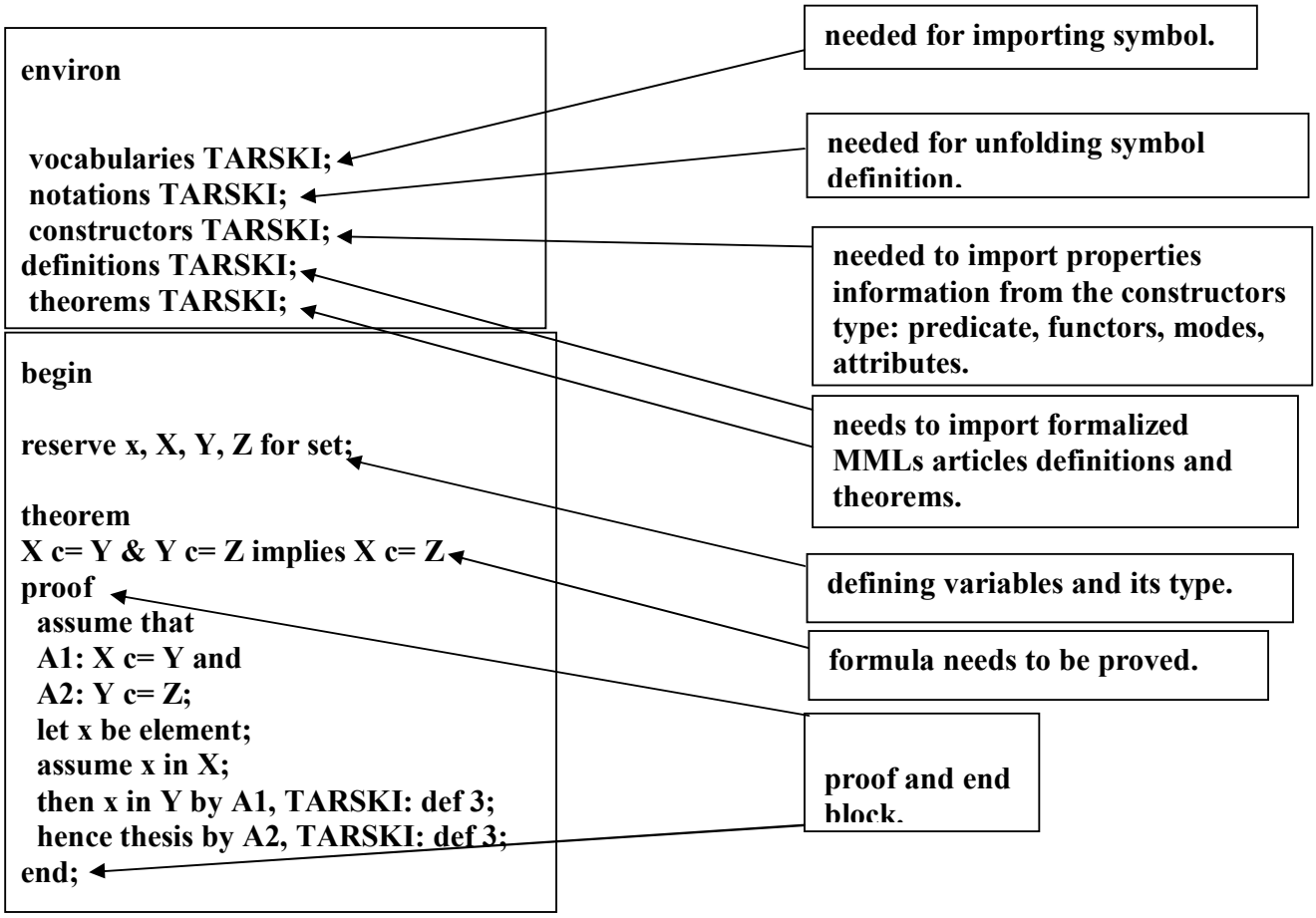

Figure 7.1. Sample of Mizar code

Here symbol c= stands for a Subset. The label A1 and A2 of local statement in a proof block is required to justify another statement written down. TARSKI: def 3 is identifier of definition by which one can justify a proof step. "let x be element" statement means x is an arbitrary object. Similarly, another popular conventional set symbol like ∩, ∪, ⊆, ∈, etc in Mizar is denoted by /\, \/, c=, in, etc.

4. Mizar Article Related to Petri net in Mizar Library

Several studies related to the formalization of the concept of Petri nets (like the basic structure of Petri nets [8], Processes in Petri nets [13], Some Elementary Notions of the Theory of Petri Nets [10], Boolean marking of Petri nets [14]) are formalized in Mizar system and are stored in the library. By using some basic Petri net knowledge from article PETRI [8], and

NET_1 [10], we built the structure of decision free Petri net and proved the token boundedness property in a circuit of the net.

Some formalized knowledge related to Petri nets concepts that we retrieved from the Mizar library in our work is described below,

A. Mizar definition of basic Petri net structure

The formalized Mizar description of the PT-net structure discussed in Definition 1 of Section 3.1 (Appendix B) is stored in the Mizar library in [8] as,

```
definition
 struct (2-sorted) PT_net_Str (# carrier, carrier' -> set,
     S-T_Arcs -> Relation of the carrier, the carrier',
     T-S_Arcs -> Relation of the carrier', the carrier #);
end;
```

Here, the Petri net structure named "PT_net_Str" consists of four selectors: the carrier, the carrier', the S-T_Arcs and the T-S_Arcs, where "carrier" represents the set of places, "carrier' " represents the set of transitions, "S-T_Arcs" represents the arc relations from places to transitions, "T-S_Arcs" represents the arc relations from transitions to places.

B. Carrier and carrier' of Petri net structure are disjoint

Another knowledge that we retrieved from NET_1 article is that elements in carrier and carrier' are disjoint, which is defined and stored in the Mizar library as:

```
definition
 let N be PT_net_Str;
 attr N is Petri means
  the carrier of N misses the carrier' of N;
end;
```

5. Mizar notations of mathematical symbols

The table below contains some Mizar notations of mathematical symbols that are used while reasoning of our work.

f /^ n	sequence f with the first n elements removed	
<=	less than or equal to	$\leq$
>=	greater than or equal to	$\geq$
=	equality	=
< >	inequality	$\neq$
f.x	the value of function f when evaluated at argument x	f(x)
{ }	empty set	null or Ø
in	a member of	$\in$
p1 ^ p2	concatenation of sequences p1 and p2	p1 ^ p2
p | n	sequence p restricted to the first n elements	
<*1,2*>	sequence made of elements 1 and 2	
dom P	domain of relation P	D_P
rng P	range of relation P	Y_P
len p	length of the finite sequence p	|p|
[:A, B:]	Cartesian product of sets	A x B
[a, b]	Kuratowski ordered pair	(a, b)
A \ / B	set theoretical union	A ∪ B
A / \ B	set theoretical intersection	A ∩ B
A c= B	A is a subset of set B	A ⊆ B

Table 2. Mizar notations of mathematical symbols

Appendix D Definitions and theorems of this work without proofs

This section contains all definitions, theorems, lemmas, etc required to prove the token boundedness property in a circuit of decision free Petri net. All these definitions, theorems, lemmas are available here without proofs.

```
begin :: Preliminaries

reserve N for PT_net_Str, PTN for Petri_net, i for Nat;

theorem Th10:
  for x,y be Element of NAT, f be FinSequence st
  f/^1 is one-to-one & 1 < x & x <= len f & 1 < y  & y <= len f & f.x = f.y
  holds x = y;

theorem Lm1:
  for D be non empty set, f be non empty FinSequence of D
  st f is circular holds f.1 = f.len f;

registration
  let D be non empty set;
  let a,b be Element of D;
  cluster <*a,b,a*> -> circular for FinSequence of D;
  coherence;
end

theorem Th13:
  for a,b be set st a <> b holds <*a,b,a*> is almost-one-to-one;

definition
  let X be set;
```

```
  let Y be non empty set;
  let P be finite Subset of X;
  let M0 be Function of X,Y;
  mode Enumeration of M0, P -> FinSequence of Y means
  :Def12:
  len it = len (the Enumeration of P) &
  for i st i in dom it holds
  it.i = M0.((the Enumeration of P).i) if P is non empty
  otherwise it = <*>Y;
end;

definition
  func PetriNet -> Petri_net equals
  PT_net_Str(# {0},{1},[#] ({0},{1}),[#] ({1},{0}) #);
  coherence;
end;

definition
  let N;
  func places_and_trans_of N -> set equals
  ((the carrier of N) \/ (the carrier' of N));
  correctness;
end;

registration
  let PTN;
  cluster places_and_trans_of PTN -> non empty;
  coherence;
end;

reserve fs for FinSequence of places_and_trans_of PTN;
```

```
definition
  let PTN,fs;
  func places_of fs -> finite Subset of the carrier of PTN equals
  {p where p is place of PTN : p in rng fs };
  coherence;
end;

definition
  let PTN,fs;
  func transitions_of fs -> finite Subset of the carrier' of PTN equals
  {t where t is transition of PTN: t in rng fs};
  coherence;
end;

begin :: The number of tokens in a circuit

::Relation Between P and NAT:::
definition
  let N;
  func nat_marks_of N -> FUNCTION_DOMAIN of the carrier of N, NAT
  equals Funcs(the carrier of N, NAT);
  correctness;
end;

definition
  let N;
  mode marking of N is Element of nat_marks_of N;
end;

:: Generation of nat marking and summation of that
```

```
definition
  let N;
  let P be finite Subset of the carrier of N;
  let M0 be marking of N;
  func num_marks(P, M0) -> Element of NAT equals
  Sum (the Enumeration of M0,P);
  coherence;
end;

begin
:: Decision-Free Petri Net Concept and Properties of Circuits in Petri Nets

definition
  let IT be Petri_net;
  attr IT is decision_free_like means
  :Def1:
  for s being place of IT holds
  ((ex t being transition of IT st [t, s]
  is Element of the T-S_Arcs of IT) &
  (for t1, t2 being transition of IT
  st [t1, s] is Element of the T-S_Arcs of IT &
  [t2, s] is Element of the T-S_Arcs of
  IT holds t1 = t2))  &
  ((ex t being transition of IT st
  [s, t] is Element of the S-T_Arcs of IT) &
  (for t1, t2 being transition of IT
  st [s, t1] is Element of the S-T_Arcs of IT &
  [s, t2] is Element of the S-T_Arcs of
  IT holds t1 = t2));
end;
:: directed_path_like Attribute for
```

```
:: FinSequence of places_and_trans_of Petri_net

definition
  let PTN;
  let IT be FinSequence of places_and_trans_of PTN;
  attr IT is directed_path_like means
  :Def5:
  len IT >= 3  & len IT mod 2 = 1 &
  (for i st i mod 2 = 1 & i + 1 < len IT holds
  [IT.i, IT.(i+1)] in (the S-T_Arcs of PTN) &
  [IT.(i+1),IT.(i+2)] in (the T-S_Arcs of PTN))
  & IT.len IT in (the carrier of PTN);
end;

theorem Th12:
  for fs be FinSequence of places_and_trans_of PetriNet st fs = <*0,1,0*>
  holds fs is directed_path_like;

registration
  let PTN;
  cluster directed_path_like -> non empty
  for FinSequence of places_and_trans_of PTN;
  coherence;
end;

:: With_directed_path Mode for place\transition Nets
definition
  let IT be Petri_net;
  attr IT is With_directed_path means
:Def9:
 ex fs being FinSequence of places_and_trans_of IT st fs is directed_path_like;
```

```
end;

:: directed_circuit_like Attribute for FinSequence of
::  places_and_trans_of PetriNet

definition
  let PTN;
  attr PTN is With_directed_circuit means
  :Def7:
  ex fs st fs is directed_path_like & fs is circular & fs is almost-one-to-one;
end;

registration
  cluster PetriNet -> decision_free_like With_directed_circuit Petri;
  coherence;
end;

registration
  cluster With_directed_circuit Petri decision_free_like for Petri_net;
  existence
  proof
    take PetriNet;
    thus thesis;
  end;
end;

registration
  cluster With_directed_circuit -> With_directed_path for Petri_net;
  coherence;
end;
registration
```

```
  cluster With_directed_path for Petri_net;
  existence
  proof
    take PetriNet;
    thus thesis;
  end;
end;

reserve Dftn for With_directed_path Petri_net;

registration
  let Dftn;
  cluster directed_path_like for FinSequence of places_and_trans_of Dftn;
  existence;
end;

reserve dct for directed_path_like FinSequence of places_and_trans_of Dftn;

theorem Thd:
  [dct.1, dct.2] in the S-T_Arcs of Dftn;

theorem The:
  [dct.(len dct -1), dct.(len dct)] in the T-S_Arcs of Dftn;

reserve Dftn for With_directed_path Petri Petri_net,
  dct for directed_path_like FinSequence of places_and_trans_of Dftn;

theorem Thc:
  dct.i in places_of dct & i in dom dct implies i mod 2 = 1;

theorem Thcc:
```

```
  dct.i in transitions_of dct & i in dom dct implies i mod 2 = 0;

theorem Thf:
  dct.i in transitions_of dct & i in dom dct implies
  [dct.(i-1),dct.i] in the S-T_Arcs of Dftn
  & [dct.i,dct.(i+1)] in the T-S_Arcs of Dftn;

theorem Thg:
  dct.i in places_of dct & 1 < i & i < len dct implies
  [dct.(i-2),dct.(i-1)] in the S-T_Arcs of Dftn
  & [dct.(i-1),dct.i] in the T-S_Arcs of Dftn
  & [dct.i,dct.(i+1)] in the S-T_Arcs of Dftn
  & [dct.(i+1),dct.(i+2)] in the T-S_Arcs of Dftn & 3 <= i;

begin :: Firable and Firing Conditions for Transitions with natural marking

reserve M0 for marking of PTN,
      t for transition of PTN,
      Q,Q1 for FinSequence of the carrier' of PTN;

definition
  let PTN,M0,t;
  pred t is_firable_at M0 means
  for m being Nat holds m in M0.:*'{t} implies m > 0 ;
end;

notation
  let PTN,M0,t;
  antonym t is_not_firable_at M0 for t is_firable_at M0;
end;
definition
```

```
  let PTN,M0,t;
  func Firing(t, M0) -> marking of PTN means
  :Defl1:
  for s being place of PTN holds
  (s in *'{t} & not s in {t}*' implies it.s = M0.s -1)  &
  (s in {t}*' & not s in *'{t} implies it.s = M0.s + 1) &
  ((s in *'{t} & s in {t}*') or (not s in *'{t} & not s in  {t}*')
  implies it.s = M0.s) if t is_firable_at M0
  otherwise it = M0;
end;

definition
  let PTN,M0,Q;
  pred Q is_firable_at M0 means
  Q = {} or
  ex M being FinSequence of nat_marks_of PTN st len Q = len M &
  Q/.1 is_firable_at M0 & M/.1 = Firing(Q/.1 , M0) &
  for i st i < len Q & i > 0 holds
  Q/.(i+1) is_firable_at M/.i & M/.(i+1) = Firing(Q/.(i+1), M/.i);
end;

notation
  let PTN,M0,Q;
  antonym Q is_not_firable_at M0 for Q is_firable_at M0;
end;

definition
  let PTN,M0,Q;
  func Firing(Q, M0) -> marking of PTN means
  :Defb:
  it = M0 if Q = {}
```

```
  otherwise
  ex M being FinSequence of nat_marks_of PTN st len Q = len M & it = M/.len M
  & M/.1 = Firing(Q/.1, M0)
  & for i st i < len Q & i > 0 holds M/.(i+1) = Firing(Q/.(i+1),M/.i);
end;

theorem
  Firing(t, M0) = Firing(<*t*>, M0);

theorem
  t is_firable_at M0 iff <*t*> is_firable_at M0;

theorem
  Firing(Q^Q1, M0) = Firing(Q1, Firing(Q,M0));

theorem
  Q^Q1 is_firable_at M0
  implies Q1 is_firable_at Firing(Q, M0) & Q is_firable_at M0;

begin
:: The theorem stating that the number of tokens in a circuit
:: remains the same after any firing sequence

theorem Thb:
  for Dftn being With_directed_path Petri decision_free_like Petri_net,
  dct being directed_path_like FinSequence of places_and_trans_of Dftn,
  t being transition of Dftn st dct is circular
  & ex p1 being place of Dftn st p1 in places_of dct &
  ([p1, t] in the S-T_Arcs of Dftn or [t, p1] in the T-S_Arcs of Dftn)
  holds t in transitions_of dct;
```

```
definition
  mode Decision_free_PT is
  With_directed_circuit Petri decision_free_like Petri_net;
  correctness;
end;

registration
  let Dftn be With_directed_circuit Petri_net;
  cluster directed_path_like circular almost-one-to-one
  for FinSequence of places_and_trans_of Dftn;
  existence;
end;

definition
  let Dftn be With_directed_circuit Petri_net;
  mode Circuit_of_places_and_trans of Dftn is
  directed_path_like circular almost-one-to-one
  FinSequence of places_and_trans_of Dftn;
  correctness;
end;

theorem Th7:
  for Dftn being Decision_free_PT,
  dct being Circuit_of_places_and_trans of Dftn,
  M0 being marking of Dftn, t being transition of Dftn
  holds num_marks(places_of dct, M0) = num_marks(places_of dct, Firing(t, M0));

theorem
  for Dftn being Decision_free_PT,
  dct being Circuit_of_places_and_trans of Dftn,
  M0 being marking of Dftn, Q being FinSequence of the carrier' of Dftn holds
```

num_marks(places_of dct, M0) = num_marks(places_of dct, Firing(Q, M0));

References

[1] J. Wang, "Deterministic timed Petri nets," in *Timed Petri nets theory and application*, Kluwer Academic Publishers, DEDS9 0-7923-8270-6, 1998, pp 37-44.

[2] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled; Model Checking. The MIT Press Cambridge, Massachusetts London, England (1999).

[3] Alessandro Cimatti, Marco Roveri, Angelo Susi, Stefano Tonetta. Formalization and Validation of Safety-Critical Requirements.

[4] G. W. Leibniz. Logical papers: A Selection. Clarendon Press, Oxford, 1966. Edited and translated by G. Parkinson.

[5] Marcos Cramer aus Buenos Aires. Proof checking mathematical texts in Controlled natural language.

[6] C. R. Vick and C. V., Ramamoorthy, "Modeling and Analysis of Concurrent Systems," Ch 3 in Handbook of Software Engineering, New York, Van Nostrand, Reinhold, 1984, pp.39-63.

[7] Available http://mizar.org/

[8] P. N. Kawamoto, Y. Fuwa, Y. Nakamura. (1992, November). Basic Petri net concept. *Formalized Mathematics*. [Online]. 3(2), pp. 183-187, MML Identifier: PETRI. Available: http://www.mizar.org/JFM/Vol4/petri.abs.html.

[9] Waldemar Korczynski. (1990, August). Some Elementary Notions of the Theory of Petri Nets. [Online], Vol. 2. MML identifier: NET_1. Available: http://www.mizar.org/JFM/Vol2/net_1.miz.html.

[10] J. M. Proth and I. Minis. (1994). "Complexity of production management in a Petri net environment," Technical Research Report No. TR-94-19, Institute for System Research, The Univ. of Maryland at College park. Available: http://drum.lib.umd.edu/handle/1903/5499

[11] E. C. Clarke, J. M. Wing, Formal Methods: State of the Art and Future Directions, ACM Computing Surveys, Vol. 28, Number 4es, pp. 116-116, December 1996.

[12] Available.http://money.cnn.com/2012/08/09/technology/knight-expensive-computer-bug/

[13] Grzegorz Bancerek, Mitsuru Aoki, Akio Matsumoto, and Yasunari Shidama. (2002, December). Processes in Petri nets. Formalized Mathematics. [Online], Vol. 14. MML identifier: PNPROC_1. Available: http://www.mizar.org/JFM/Vol14/pnproc_1.abs.html.

[14] Pauline N. Kawamoto, Yasushi Fuwa and Yatsuka Nakamura. (1993, October). Basic

Concepts for Petri Nets with Boolean Markings. Formalized Mathematics. [Online] , Vol. 5. MML Identifier: BOOLMARK. Available: http://www.mizar.org/JFM/pdf/boolmark.pdf.

[15] T. Murata. (1989, Apr). Petri nets: Properties, analysis and applications. *Proceeding of the IEEE,* vol. 77, no. 4, pp.541-580. Available: http://embedded.eecs.berkeley.edu/Respep/Research/hsc/class.F03/ee249/discussionpapers/PetriNets.pdf

[16] Shah, Pratima K., Kawamoto, Pauline N., and Giero, Mariusz. The Formalization of Decision Free Petri Net. FORMALIZED MATHEMATICS, Vol. 22, No. 1, pp. 29–35 (2014). Online: http://fm.mizar.org/fm22-1/petri_df.pdf.

[17] file:///C:/Users/Pauline%20Kawamoto/Downloads/PETRI_DF%20(1).htm

[18] Matuszewski R, Rudnicki P 2005 Mizar: the first 30 years, Mechanized mathematics and its applications 4(1): 3–24.

[19] A. Trybulec. (1996). On the Decomposition of Finite Sequences. *Formalized Mathematics*. 5(3), pp. 317-322, MML Identifier: FINSEQ_6.

[20] Robert Milewski. Subsequences of Almost, Weakly and Poorly One-to-one Finite Sequences. Formalized Mathematics, Vol. 13, No. 2, pp. 227-233, 2005. MML Identifier: JORDAN23.

[21] E. C. Clarke, J. M. Wing, Formal Methods: State of the Art and Future Directions, ACM Computing Surveys, Vol. 28, Number 4es, pp. 116-116, December 1996.

[22] A. Harry, Formal Methods Fact File: VDM and Z, Wiley, 1996

[23] Information Available: http://www.airfrance447.com/.

[24] Martyn Thomas. Knight capital software upgrade costs $440m. http://catless.ncl.ac.uk/Risks/, August 2012. The Risks Digest Forum on Risks to the Public in Computers and Related Systems.

[25] Available http://en.wikipedia.org/wiki/Computerized_system_validation

[26]Availavle http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/

[27] Kaufmann, M., Moore, J.S.: Some Key Research Problems in Automated Theorem Proving for Hardware and Software Verification. In: Spanish Royal Academy of Science (RAMSAC). Volume 98. (2004) 181–196

[28] Brock, B.C., Hunt, W.A.: Formally Specifying and Mechanically Verifying Programs for the Motorola Complex Arithmetic Processor DSP. In: Proceedings of the IEEE International Conference on Computer Design (ICCD'97). (1997) 31–36.

[29] Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petruccci, L., Schnoebelen, P., McKenzie, P.: Systems and Software Verification. Springer-Verlag (2001)

[30] D. Boˇsnaˇcki, D. Dams, L. Holenderski, Symmetric Spin, 7th Int. SPIN Workshop on Model Checking of Software SPIN 2000, pp. 1-19, LNCS 1885, Springer, 2000.

[31] Pnueli, A., 81. "A Temporal Logic of Concurrent Programs," Theoretical Computer Science, vol. 13, pp. 45-60, 1981.

[32] Vardi, M.Y. and Wolper, P., 86. "An automata-theoretic approach to automatic program verification," in Proc. of the 1st Symposium on Logic in Computer Science, Cambridge, June 1986, pp. 322-331.

[33] Fernandez, J.C., Garavel, H., Kerbrat, A., Mateesku, R., Mounier, L., and Sighireanu, M., 96. "CADP: A Protocol Validation and Verification Toolbox," in Proc. of the 8th International Conference on Computer-Aided Verification (CAV'96), New Brunswick, NJ,USA, July/August 1996. Lecture Notes in Computer Science 1102, pp. 437-440.

[34] Fernandez, J.C., Garavel, H., Mounier, L., Rasse, A., Rodriguez, C., and Sifakis, J., 92b. "A Toolbox for the Verification of LOTOS Programs," in Proc. of the 14th International Conference on Software Engineering (ICSE'14), Melbourne, Australia, May 1992, pp. 246-259. L. A. Clarke, Ed.

[35] http://hol.sourceforge.net/

[36] Hales, T. (2005). Introduction to the Flyspeck project. Mathematics, Algorithms, Proofs.

[37] Matuszewski, R., & Rudnicki, P. (2005). Mizar: the first 30 years. Mechanized Mathematics and Its Applications, 4, 3–24.

[38] R. Milner. (1980). A calculus of communicating systems. vol. 92 of Lecture Notes in Computer Science. Springer-verlag.

[39] J. Bechta-Dugan and K. S. Trivedi, "Coverage modeling for dependability analysis of fault-tolerant systems," IEEE Trans. Comput., vol. 38, no. 6, pp. 775-787. 1989.

[40] F. Belli and K.-E. Grosspietsch, "Specification of fault tolerant systems issues by predicate/transition nets and regular expressions-approach and case study," IEEE Trans. software Eng.. vol. 17, no. 6, pp. 513-526, 1991.

[41] S. Jorg, Invariance properties in Distributed Systems, Institut Fur Informatik, Universitat Kiel, W-2300 Kiel.

[42] F. Commoner, A. W. Holt, S. Even, and A. Pnueli. (1971). Marked directed graphs. *journal of Computer and System Sciences, vol. 5*, pp. 511-523.

[43] Fitch F B 1952 Symbolic Logic, An Introduction, The Ronald Press Company.

[44] Cheung, K. S. (2004) New characterization for live and reversible augmented Petri nets. Information Processing Letters, vol.92, no.5, pp239-243.

[45] Chu, F., Xie, X.L. (1997) Deadlock analysis of Petri nets using siphons and mathematical programming. IEEE Transactions on Robotics and Automation, vol.13, no.6, pp.793-804.

[46] Available http://en.wikipedia.org/wiki/Computerized_system_validation