Department of Computer Science
Aalborg University
Fredrik Bajersvej 7E
DK–9220 Aalborg Øst
Denmark

Automatic Translation of Timed-Arc Petri Nets to Timed Automata

Master's Thesis
Aalborg University
June 2005

Group members: Krishna Prasad Gundam | gundam@cs.aau.dk
Ye Tian | tianye@cs.aau.dk

Faculty of Engineering and Science
Aalborg University

Department of Computer Science

TITLE:

Automatic Translation
of Timed-Arc Petri Nets
to Timed Automata

THESIS PERIOD:
SSE4,
February – June 2005

GROUP MEMBERS:
Ye Tian
Krishna Prasad Gundam

SUPERVISOR:
Jiří Srba

NUMBER OF COPIES: 6

REPORT PAGES: 76

APPENDIX PAGES: 6

TOTAL PAGES: 91

SYNOPSIS:

The main objective of this thesis is to investigate possibilities of automatic verification of TAPN. We focus on translations of various subclasses of TAPN into TA. Our aim is to verify the reachability properties of TAPN by verifying the translated TA using verification tool UPPAAL.

Our reduction technique works for a sub-class of TAPN namely k-conservative nets. And the translator is implemented on Java platform considering both input and output files as XML files. Moreover we present two case studies on Fischer's protocol and Alternating Bit Protocol.

Acknowledgement

Our heartfelt gratitude goes first to our mentor Jiří Srba, whose academic guidance and illuminating suggestions have encouraged and helped us greatly in our master degree study and in the completion of our thesis. We are very grateful for his advise and valuable discussions.

Our warm thanks go also to the Umbrella group and Morten Kühnrich for their help with LaTeXwriting.

To My parents, *Guoqing Tian* and *Xiuzhen Li*
My love, *Lillian*

Ye Tian

To My mother, *Vijaya Kumari Gundam*, father, *Narayana Reddy Gundam*
My family and friends

Krishna Prasad Gundam

Contents

Contents **7**

1 Introduction **9**
- 1.1 Technology Elevation . . . 9
- 1.2 Formal Methods . . . 10
- 1.3 Our Focus . . . 12
- 1.4 Related Work . . . 13
- 1.5 Report Overview . . . 13

2 Basic Definitions **15**
- 2.1 Petri Nets . . . 15
- 2.2 Timed Automata . . . 27

3 Translations from TAPN to TA **33**
- 3.1 Basic Idea of Translation . . . 33
- 3.2 Removing Useless Transitions . . . 33
- 3.3 Reduction of Degree of TAPN . . . 36
- 3.4 Reduction from TAPN to TA/TAN . . . 45
- 3.5 Conclusion . . . 56

4 Implementation **59**
- 4.1 Introduction . . . 59
- 4.2 Tools and Programming Languages Used . . . 59
- 4.3 XML Reader and Writer . . . 61
- 4.4 Reduction Algorithms . . . 65

5 Case Study **67**
- 5.1 Case Study I: Fischer's Protocol . . . 67
- 5.2 Case Study II: Alternating Bit Protocol . . . 72

6 Conclusion and Future Work **81**

6.1 Our Contribution 81
6.2 Future Work 82

A Tools and Howtos **83**
A.1 Roméo 83
A.2 UPPAAL 83
A.3 XMLSpy 84

B DTD **85**
B.1 Analysis and Diagram 85
B.2 DTD Code 86

C CD Packages **87**
C.1 Readme 87
C.2 Tools 87
C.3 Program Packages 87
C.4 Examples 88

Bibliography **89**

1 Introduction

This report is a documentation of our master thesis, which specifies about the translation techniques from Timed Arc Petri Nets into Timed Automata. The report signifies both formal proof of the translations and the practical implementation of the achieved results. The report is made as part of our masters thesis at Department of Computer Science, Aalborg University, Denmark.

1.1 Technology Elevation

In this age of technology, mankind lifestyle is greatly influenced by the latest technological advancements. Of all the inventions mankind has made so far, invention of computers has vital role because of its widespread usage in many activities ranging from mail sending to missile launching. Areas of computer applications have found no boundaries and they became integral part of human life. The areas of its applications also became wider. Even home appliances as small as coffee machines, T.V. remote controllers have computers in them. As we further narrow down the areas of computers applications, important machines like nuclear power plants station shutdown machines, aerospace machines and medical equipment are controlled by computers.

Computer programs which are embedded in these kind of systems are called as *Embedded Systems* and those which react to external stimuli are *Reactive Systems*.

1.1.1 Embedded and Real-Time Systems

Definition of **Embedded System** as given in *Wikipedia* [40] encyclopedia:

An Embedded System is a special-purpose computer system, which is completely encapsulated by the device it controls. An embedded system has specific requirements and performs pre-defined tasks, unlike a general-purpose personal computer.

In *Wikipedia* encyclopedia, we have the definition of

Real Time System

A Real-Time System responds in a (timely) predictable way to unpredictable external stimuli arrivals. In short, a Real-Time System has to fulfil under extreme load conditions.

These real time systems or reactive systems are prone to change in their behavior and also react to changes in real time. Obviously correctness of these systems is very crucial as their areas of applications are hazard prone. Failure of such systems may result in catastrophe, hence design of such systems as reliable ensuring safety is vital.

1.1.2 How Secured Are We?

As we can see various applications of these systems gave numerous positive implications to humans, at the same time one should not over look the potential threats that may be caused because of these systems. Stopping usage of such systems, is not a wise solution to the problem. Instead it is better to design these systems as failure proof thereby ensuring their services to mankind. There is growing concern about the safe usage of such systems considering their potential threat under failure circumstances.

Thoroughly tested and software usage in these systems in not merely a solution instead the whole process of verifying such systems is mechanized. So here the computers again come into rescue of verifying such systems which is termed as computer aided verification. In the following section formal methods employed to ensure the security of these systems are explained.

1.2 Formal Methods

As defined in [40], formal methods refer to mathematically based techniques for the specification, development and verification of software and hardware systems.

Formal methods can be applied to specify the system and also can be used as guide to develop concrete system. The other main feature of formal methods is to verify once a formal specification has been developed. Possible approaches of the formal methods are

1. Human-Directed Proof, and
2. Model checking, in which a system verifies certain properties by means of an exhaustive search of all possible states that a system could enter during its execution.

In the following sections the model checking is explained in detail.

1.2.1 Model Checking

Quoted from [40], **Model Checking** *is a method to algorithmically verify finite state systems formally. This is achieved by verifying if the model, often deriving from a hardware or software design, satisfies a logical specification.*

In the model checking phenomenon the specification is written as temporal logic formulas and the model is expressed as a state transition system. The transitions systems are the graphical representation of these systems with nodes and edges. They are directed graphs consisting of nodes (or vertices) and edges. The nodes represents states of a system, edges represent execution of the system which alters the state of the system enabling it moving from one node to the other.

In the following section the formal definition of labeled transition system is given.

1.2.2 Labeled Transition System

First the definition of transition is presented and is followed by the various extensions to these transition systems.

Definition 1. *A* **transition system** *is a pair of the form*

$$(S, \rightarrow)$$

where

1. *S is a set of states (configurations), and*
2. *$\rightarrow$ is a set of transitions (transition relations) such that $\rightarrow \subseteq S \times S$.*

Basically a transition system only defines the states and transitions themselves, i.e., if the system can move from one state to another. But it does not care much about the distinguishing qualities among each transitions and how the system moves among states. In further discussion, we need stronger constraint of transitions and introduce a system which is the fundamental research object in Petri nets and timed automata, called *labeled transition system* (called *LTS* in short).

The LTS is the extend notion of the transition system. A LTS consists of a collection of states and a collection of transitions between them. The transitions are labeled by actions that happen when the transition is taken (or fired).

Formally, it is defined as:

Definition 2. *A* **labeled transition system** *is a triple*

$$(S, \mathcal{L}, \rightarrow)$$

where

1. *S is a set of states (configurations),*
2. *$\mathcal{L}$ is a set of labels, and*
3. *$\rightarrow$ is a set of transitions $\rightarrow \subseteq S \times \mathcal{L} \times S$, and if $(S_1, \alpha, S_2) \in \rightarrow$ then we write $S_1 \xrightarrow{\alpha} S_2$.*

Figure 1.1 illustrate a labeled transition system which shows a simple gear operation. *Stop* is the initial state, and with a transition `ShiftGearUP`, it goes to node *Gear1*, i.e., it runs at a low speed and could shift to either *Stop* (through the transition `ShiftGearDown`) or *Gear2* (through the transition `ShiftGearDown`) which has a higher speed. When it reaches the state *Gear2*, the system can only go back to node *Gear1* (through the transition `ShiftGearDown`).

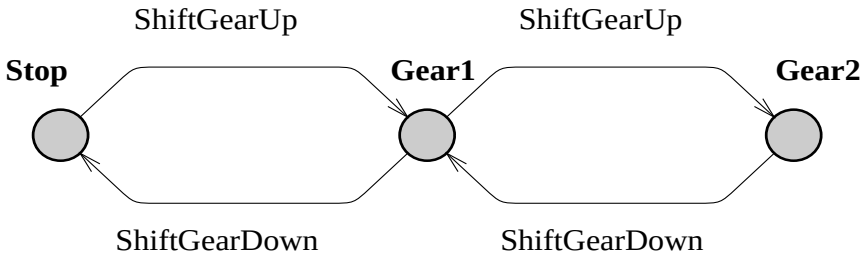


Figure 1.1: An Example of LTS

These labeled transition systems are extended with time and many other features so that the real time or safety critical systems can be modeled according to their behavior. Timed automata and Petri nets falls into such category. The detailed definitions of these are explained clearly in following chapters.

1.3 Our Focus

We are working on one of the subclasses of Petri Nets with time called conservative Timed-Arc Petri Nets. Timed-arc Petri nets are useful in modeling various real world examples and by doing so we can effectively check several systems properties. The class of nets we are dealing with are the nets where time constraints are associated to arcs, and this class is called Timed-Arc Petri Nets (TAPN). In this class, tokens in each place are of certain ages where the tokens with correct age are used for firing the transitions.

One of the most popular ways of modeling real time system is to model it using the Timed Automata where the systems can be modeled according to clock conditions or constraints based on time. This timing aspect in modeling brings it very close to real world systems like traffic lights, and many embedded controllers. Most of our work is related with the timed-arc Petri nets (TAPN) and timed automata (TA).

1.3.1 Motivation Behind Our Work

We know how important and how useful it is if we model the real world systems. There are several methods and ready made modeling tools for doing it so. For example when we model the system as TA we have tools like Uppaal [37], KRONOS [24] etc., and if we want to model the system as Petri nets (PN) and accordingly its extensions then for colored PN we have CPN tools, for Timed PN Roméo [32] and Tina [36]. But to our knowledge there is no tool available so far for modeling the TAPN.

Since there is no tool to model and verify the TAPN we are interested in finding a solution for doing it. One of the ways we have chosen is to convert the TAPN to the TA as the TA has many verification tools. Here the problem would be to develop an algorithm such that it can convert the TAPN into TA.

1.4 Related Work

The related work on converting these Petri nets into timed automata is explained in the papers [13]. In this paper the translation of Timed Petri nets into timed automata is explained. Their model is, however, different from TAPN and the reduction idea is also different.

1.5 Report Overview

The rest of the report is organized as follows. Chapter 2 contains the formal definitions of all the classes of Petri nets we have mentioned throughout the report. We also gave brief description of the timed automata along with introduction to the Uppaal tool.

Chapter 3 includes the reduction techniques we have proposed to reduce the TAPN into TA. Chapter consists all the proof results and techniques of some of the classes of TAPN.

Chapter 4 gives insight knowledge about practical implementation of the reductions we have presented in earlier chapters. Algorithms are implemented

and explained.

Chapter 5 documents the experiments we did on two of the case-studies we studied namely Fischer's protocol and Alternating Bit Protocol.

Chapter 6 concludes the report enlightening future enhancements to our work.

Appendices A and B contain the practical information about how to install the tools we are using, and DTD of the TAPN we designed. Appendices C and D contain the parser codes and example of input/output XML files. Appendices E refers to CD package along with the Readme file and program packages with examples.

2 Basic Definitions

In this chapter we will describe the basic definitions of Petri Nets, Automata and timed extensions to them. Also chapter gives introduction about the various problems of the Petri Nets like reachability property, coverability property, liveness property and deadlock property. Apart from the above topics we will give brief introduction to the UPPAAL tool.

2.1 Petri Nets

Petri nets were devised in 1962 by Carl Adam Petri [31], as a tool for modeling and analyzing processes. One of the strengths of this tool is the fact that it enables processes to be described graphically. Despite the fact that Petri nets are graphical, they have a strong mathematical basis. Unlike many other schematic techniques, they are entirely formalized. It is often possible to make strong statements about the properties of the process being modeled. There are also several analysis techniques and tools available which can be applied to analyze a given Petri net.

Over the years, the model proposed by Carl Adam Petri has been expanded upon in many different ways. It is possible to model complex processes in an accessible way.

Let us see various extensions of the Petri Nets. The three most important extensions are:

a) colored extension,

b) timed extension, and

c) hierarchical extension.

We call Petri nets extended with color, time, and hierarchy high-level Petri nets.

In this report we focus on the timed extension to these Petri Nets.

2.1.1 Ordinary Petri Nets

Definition 3. *(Place/Transition Nets) A* **Place/Transition Net***, or just a* **Petri Net***, is a four tuple $N = (P, T, F, M_0)$ such that*

1. *P is a finite set of* **places***, T is a finite set of* **transitions** *$(P \bigcap T = \emptyset)$.*
2. *F is called the* **flow relation***, $F \subseteq (P \times T) \bigcup (T \times P)$.*
3. *$M_0 : P \to \mathbb{N}$; M_0 is called the* **initial marking** *of N; in general, a mapping $M : P \to \mathbb{N}$ is called a* **marking** *of N.*

Given $a \in P \bigcup T$, the *preset* of a, denoted by ${}^\bullet a$, is defined as

$$\{a' \mid (a', a) \in F\};$$

while the *postset* of a, denoted by $a^\bullet$, is defined as

$$\{a' \mid (a, a') \in F\}.$$

Definition 4. *(Firing Rule of P/T Nets) Firing rule of a P/T net is the rule under which condition a transition can be fired. It shows the way how an ordinary P/T net runs.*

Let $N = (P, T, F, M_0)$ be a P/T net, and M is a marking in it, and $t \in T$.

1. *We say that t is* **enabled** *at marking M if and only if:*

   $$\forall p \in {}^\bullet t. M(p) > 0$$

   i.e., on each precondition of t we have at least one token.

2. *If t is enabled at M, it can be fired, and by its firing we reach a marking M' which is obtained, for every place p, as follows:*

   $$M'(p) = M(p) + F(t, p) - F(p, t),$$

   where

   $$F(x, y) = \begin{cases} 1 & \text{if } (x, y) \in F \\ 0 & \text{otherwise.} \end{cases}$$

   Thus, from each precondition place of t we remove a token, and we add a new token on each postcondition place of t. As usual, we denote these evolutions by $M[t\rangle M'$.

If for a sequence of markings

$$M_0, M_1, M_2, ...M_n$$

we have

$$M_0[t_1\rangle M_1[t_2\rangle M_2[t_3\rangle \cdots [t_n\rangle M_n$$

then we say that M_n is **reachable** from M_0 and the sequence

$$\sigma = t_1, t_2, t_3, ...t_n$$

is called an **occurrence sequence**.

We denote this by $M_0[\sigma\rangle M_n$ and say that M_n is the marking **reached** via σ.

Definition 5. *(Reachable Marking) We say that a marking M of a Petri net is reachable if it is reachable by some occurrence sequence from the initial marking. The set of reachable markings of the net* (P, T, F, M_0) *is denoted by* $[M_0\rangle$.

We should note that the empty sequence is an occurrence sequence and it reaches the initial marking M_0.

A Simple Example

Here we show a simple example of Place/Transition net.

Imagine there is a very simple library system: two students A and B, and a book in this little library. The basic rule is that only one person can borrow the book from the library. Upon this problem, we give the following model to represent the system, as illustrated in Figure 2.1.

The places which have initial tokens are A, B and *Library*. This means that at the beginning, both A and B have library card and can borrow the book which is available in the library. In this case, both of the transitions A_{borrow} and B_{borrow} are enabled because there is a token in each of the places, either A and *Library*, or B and *Library*. Let us assume that transition A_{borrow} is fired at this time (if B_{borrow} is fired first, the situation is symmetric), which means A successfully borrowed the book first. Hence tokens in A and *Library* are consumed while there generate another new token in $A_{holding}$, this means that A is holding the book now. Since there is no token in *Library* any longer, we can see that B_{borrow} is not enabled now, which means B can not borrow the book temporarily. And there is only one transition A_{return} enabled in the net now. This situation is illustrated in Figure 2.2.

When A_{return} is fired, token in $A_{holding}$ is consumed while in each of the places, A and *Library*, one new token is generated, which means that A returns the book and the book is available in the library again. Thus the system goes back to its initial states, where each of A, *Library*, and B has a token, as illustrated in Figure 2.1. Again, both A_{borrow} and B_{borrow} are enabled in this case, such that both A and B have the chance to borrow the book now. If it is B who

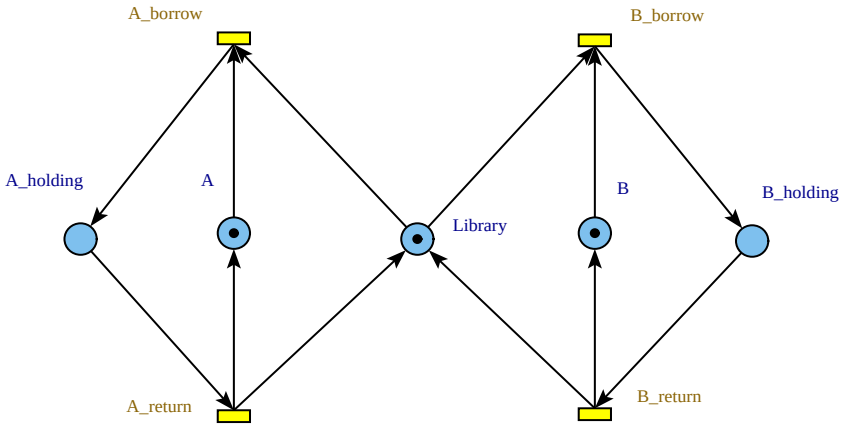


Figure 2.1: A P/T net for the Little Library System

successfully borrowed the book at this time, then the similar situation could be described as Figure 2.3.

Because there are no more constraints on this model, A and B has the same priority at this time. We can not make sure which transition will be fired next time, thus we do not know which person will borrow the book when the book is available again. We need some more conditions to consider the time features, thus we introduce a class of Petri nets with timed extension in Section 2.1.2.

In the following, we will present some subclasses of the ordinary Petri nets. Definitions can be found e.g. in the paper presented by Hee and Sidorova [21].

K-Bounded Petri Nets

Definition 6. *A marking* M *of a net* N *is said to be* **k-bounded** *if, for all places p of the net, the number of tokens in that place never exceeds k. A Petri net is* **k-bounded** *if all reachable markings are k-bounded. Such that,*

$$\forall p \in P,\ M(p) \leq k.$$

Particularly, 1-bounded nets are also called **1-safe** nets, i.e., for every place p in the net and for any reachable marking M, it holds

$$M(p) \leq 1.$$

1-safe nets have stronger constraint comparing to the k-bounded nets. Because in a 1-safe net, the 1-safe property limits each place can have only 1 token at

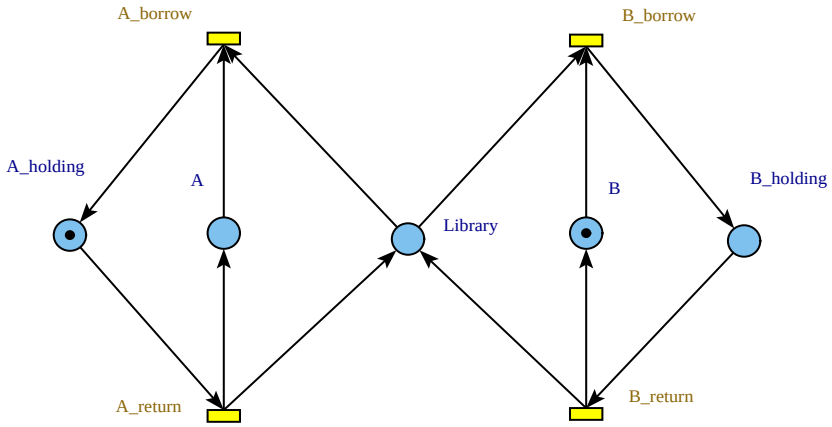


Figure 2.2: A Borrowed Book

most, thus the property limits the total number of tokens can not be greater than the number of places.

Figure 2.4 illustrates an example of 2-bounded Petri net.

K-Conservative Petri Nets

There is another class of nets, which constantly holds the same number of tokens all the time, defined as below.

Definition 7. *A net* N *is* **conservative** *if and only if all its reachable markings have exactly the same number of tokens. Thus a net* N *is* **k-conservative** *if and only if for every reachable marking* M*, we have*

$$\sum_{p \in P} M(p) = k$$

Figure 2.5 illustrates an example of 3-conservative Petri net.

2.1.2 Timed Extension of Petri Nets

Given a process modeled as a Petri net, we often want to be able to make statements about its expected performance. If we produce a model of the traffic lights at a road junction, then we are probably also interested in the number of vehicles which this junction can handle per hour. If we model the production

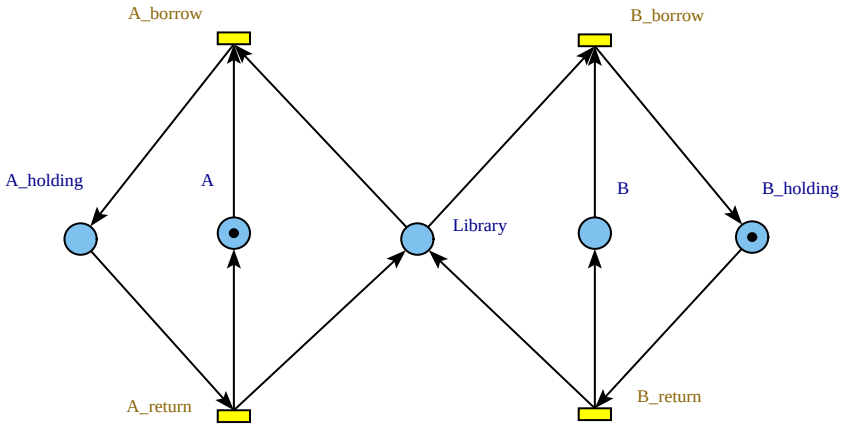


Figure 2.3: Similar Situation if B Borrowed the Book

process in a car factory, then we also want to know the expected completion time and the capacity required. In order to be able to answer these questions, it is necessary to include pertinent information about the timing of a process in the model. The classical Petri net, however, does not allow the modeling of "*time*". Even with color extension, it is still difficult to model the timing of a process. Therefore, we will extended classical Petri nets with time.

Definitions in this subsection are based on [15].

Definition 8. *(Timed-arc Petri Net) A* **Timed-Arc Petri Net** *is defined as a four tuple N=(P, T, F,* `times`*) such that,*

1. *P is a finite set of* **places**, *T is a finite set of* **transitions** $(P \cap T = \emptyset)$,
2. *F is the* **flow relation**, $F \subseteq (P \times T) \cup (T \times P)$,
3. `times` *is a function that associates to each arc (p,t) in F a pair of natural numbers, the second of which can be infinity,*
   i.e., `times`*:* $F|_{P \times T} \rightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$.

In the meanwhile, we define the terminology **size** as follows.

Definition 9. *Let* $N = (P, T, F, \texttt{times})$ *be a TAPN, the size of N is defined as* $|N| = |P| + |T| + |F|$.

To simplify some definitions we consider only arcs with weight 1, but the extension to general arcs with greater weights is straightforward.

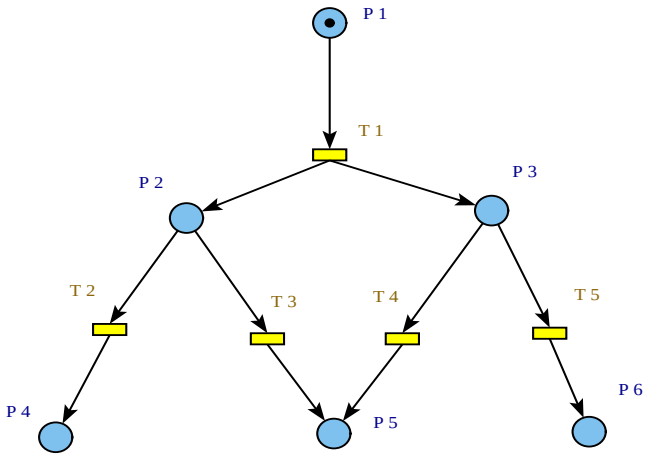


Figure 2.4: An Example of 2-Bounded Petri Net

When $\texttt{times}(p,t) = [t_1, t_2]$ we write $\pi_i(p,t)$ to denote t_i, for $i = 1, 2$. Since `times` defines the intervals of age of the tokens to be consumed by the firing of each transition (see Definition 10), we will always have $\pi_1(p,t) \leq \pi_2(p,t)$. Moreover, we will write $x \in \texttt{times}(p,t)$ to denote $\pi_1(p,t) \leq x \leq \pi_2(p,t)$.

Markings are defined by means of multisets on $\mathbb{R}^+$. Thus, a marking M is a function $M\colon P \to \mathcal{B}(\mathbb{R}^+)$ where $\mathcal{B}(\mathbb{R}^+)$ denotes the set of finite multisets of real numbers. This means each place is annotated with a certain number of tokens, and each one of them has associated a non-negative real number (its *age*). We denote the set of markings of N by $\mathcal{M}(N)$, and using classical set notation, we denote the total number of tokens on a place p by $|M(p)|$.

Definition 10. *(Firing Rule of TAPN) Firing rule of a TAPN is the rule under which condition a transition can be fired. It shows the way how a TAPN runs and it is more complicated compared to an ordinary P/T net.*

Let N=(P, T, F, `times`*) be a TAPN, and M is a marking in it, and* $t \in T$.

1. *We say that t is* **enabled** *at marking M if and only if:*

$$\forall p \in {}^\bullet t \ \exists x_p \in \mathbb{R} \text{ such that } x_p \in M(p) \wedge x_p \in \text{times}(p,t) \qquad (1)$$

   i.e., on each precondition of t we have a token whose age belongs to `times`*(p,t).*

2. *If t is enabled in M, it can be fired, and by its firing we reach a marking*

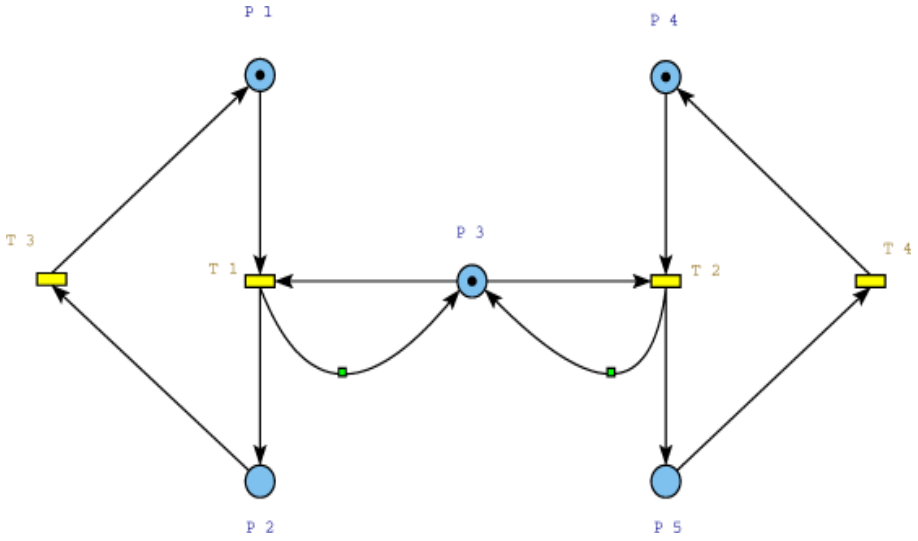


Figure 2.5: An Example of 3-Conservative Petri Net, notice that it is also 1-safe

M' which can be obtained as follows:

$$M'(p) = M(p) - C^-(p,t) + C^+(t,p),\ \forall p \in P$$

where both the subtraction and the addition operators work on multisets, and

- $C^-(p,t) = \begin{cases} \{x_p\} \text{ where } x_p \in M(p) \text{ and } x_p \in \texttt{times}(p,t) & \text{if } p \in {}^\bullet t \\ \emptyset & \text{otherwise} \end{cases}$
- $C^+(t,p) = \begin{cases} \emptyset & \text{if } p \notin t^\bullet \\ \{0\} & \text{otherwise.} \end{cases}$

Thus, from each precondition place of t we remove a token fulfilling (1), and we add a new token (with age 0) on each postcondition place of t.

As usual, we denote these evolutions by $M[t\rangle M'$, and it is noteworthy that these evolutions are in general non-deterministic, because when we fire a transition t, some of its precondition places could hold several tokens with different ages, that could be used to fire it. Besides, we see that the firing of transitions does not consume any time.

Therefore, we define another evaluation to model the passage of time as below. In this case, we increase the age of the tokens in the net by the same time:

3. *If $d \in \mathbb{R}^+$ is the passage of time in marking M and no tokens are fired, then we define*

$$\forall p \in P. M'(p) = M(p) + d$$

which means the operation on a multiset such that

$$M'(p) = \{x_p + d \mid x_p \in M(p)\}.$$

We denote this by $M[\epsilon(d)\rangle M'$, say that the age of the tokens in the net is increased by the same time d.

A Simple Example of TAPN

Let us see a simple example of TAPN. Again, we recall the *small library system* in Section 2.1.1. There is a obvious limitation in the system that we can not make sure which transition will be definitely fired when the both the two transitions, A_{borrow} and B_{borrow}, are enabled. As we cannot make every decision only by will or chance in life, we need some "*rule*" for the system. In this case, we modify it with time extensions, shown in Figure 2.6.

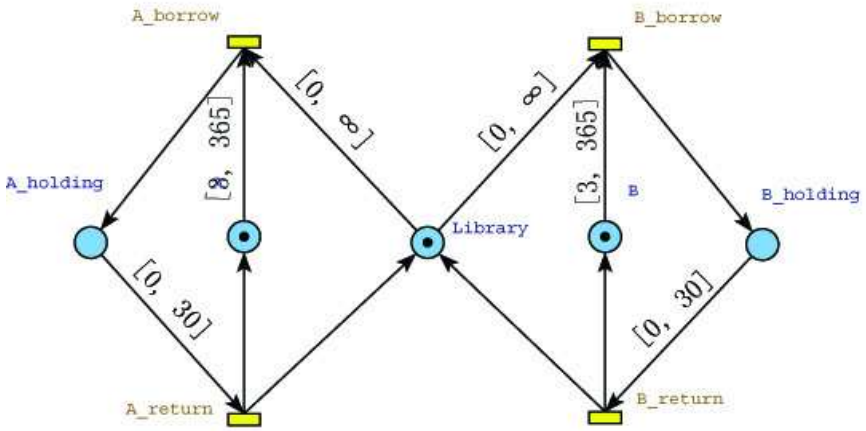


Figure 2.6: Add Time Extension to the Small Library System

The time extension in this modified system adds time constraints. For example, on the arc from place A to transition A_{borrow}, we give a time constraint $[3, 365]$, i.e., A can borrow the book from library after he gets the library card at least 3 days, and should be no later than 365 days to keep the card active and valid. And the time constraint $[0, 30]$ on the arc from place A_{borrow} to transition A_{return} means that A can validly hold the book for 30 days. If A holds the book for more than 30 days, then the transition A_{return} is not enabled any longer, i.e., the valid return procedure is not available to A any more. Thus we can active another "$paying fine$" system in this case. But we here only focus on the time extension and will not give the whole complicated system in details. The time constraint on the arcs from place *Library* to transitions A_{borrow} and B_{borrow} mean that the library could lent the book as long as it is available. Similarly, the situation of B is symmetric.

In the modified Petri net with timed extension, which is called TAPN, we can then solve the problem of deciding whom the book should be lent to. Because once the student who borrowed the book returns it, the age of the new generated token in the net is 0, i.e., it has to wait until the age increases to 3, then it has the authority to borrow a book from the library. While during this period, only the transition on the other side is enabled, i.e., the other student has a higher priority rather than this student. Thus the timed extension distinguishes them from each other.

In the following, we will present some subclasses of the timed-arc Petri nets. Definitions can be found in the paper by Hee and Sidorova [21].

k-Bounded TAPN

Definition 11. *We say that a marking M is* **k-bounded** *if*

$$\forall p \in P. |M(p)| \leq k.$$

A TAPN N is **k-bounded** *if all its reachable markings are k-bounded.*

Similar to the way in ordinary Petri nets, 1-bounded TAPN are also called 1-safe TAPN.

Figure 2.7 illustrates an example of 2-bounded TAPN.

k-Conservative TAPN

Definition 12. *A TAPN N is* **k-conservative** *if and only if all its reachable markings have exactly k tokens, i.e., for every reachable marking M, we have*

$$\sum_{p \in P} |M(p)| = k$$

Figure 2.8 illustrates an example of 3-conservative TAPN.

2.1.3 Selection of Problems

According to [31], there are two categories of problems related to Petri nets and they are *Dynamic* and *Static Problems*.

Dynamic problems characterize the behavior of individual Petri nets for example whether it is possible to reach a marking in which no step is enabled. It

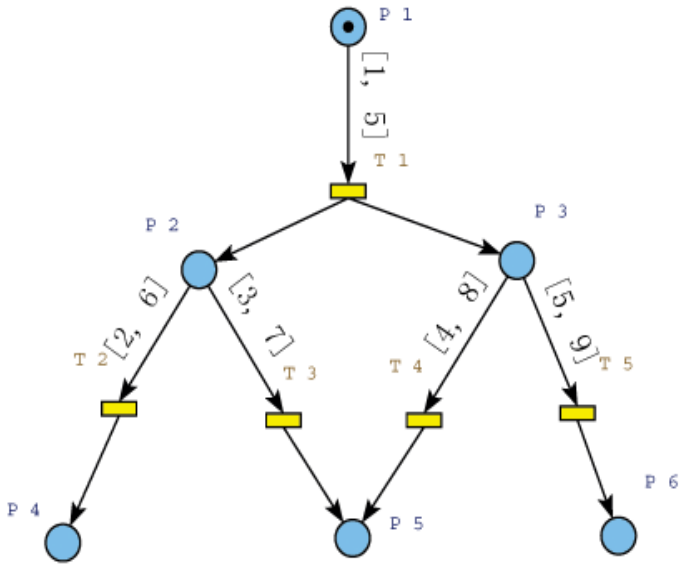


Figure 2.7: An Example of 2-Bounded TAPN

is often rather difficult to verify dynamic problems in particular when relying only on informal arguments.

Static problems can be decided from the definition of individual Petri Nets without considering the possible occurrence sequences.

Reachability Problem

The *reachability problem* for a net N (either a P/T net or a TAPN) with initial marking M_0 is the problem of deciding for a given marking M of N if it is reachable from the marking M_0, i.e., to decide if

$$M \in [M_0\rangle.$$

There are few varieties of reachability problems like

1. Submarking Reachability Problem: The Submarking reachability problem restricts the reachability problem to consider only a subset of places, not caring about the markings of other places.
2. Zero-Reachability Problem: The zero-reachability problem asks if the specific marking with zero tokens in all places is reachable.
3. Single-Space Zero Reachability Problem: The single-space zero reachability problem asks if it is possible to empty all the tokens out of a particular place.

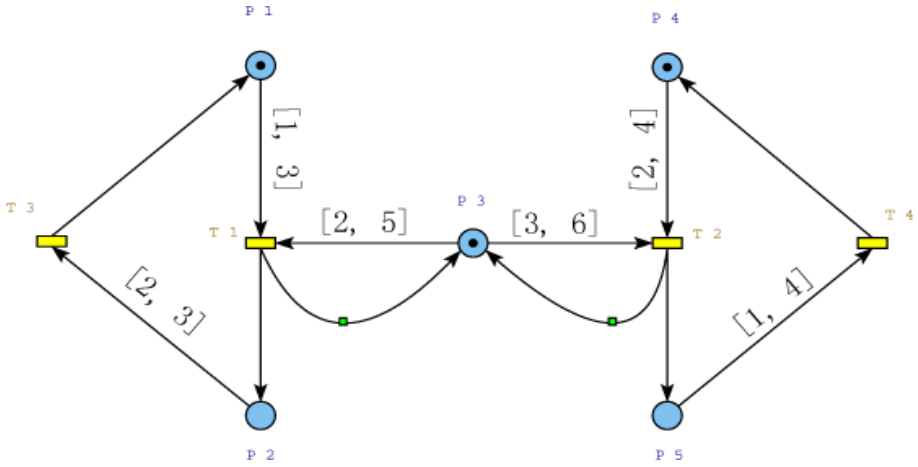


Figure 2.8: An Example of 3-Conservative TAPN, it is also 1-safe

Coverability Problem

The *coverability problem* for a TAPN N with initial marking M_0 is the problem of deciding for a marking M if there exists a marking M' where $M' \in [M_0\rangle$ such that for all places p in the net, $M(p)$ is a subset of $M'(p)$, i.e., to decide if

$$\exists M'.M' \in [M_0\rangle \wedge \forall p \in P.M(p) \subseteq M'(p).$$

While for a P/T net N with initial marking M_0 is the problem of deciding for a marking M if there exists a marking M' where $M' \in [M_0\rangle$ such that for all places p in the net, $M(p) \leq M'(p)$, i.e., to decide if

$$\exists M'.M' \in [M_0\rangle \wedge \forall p \in P.M(p) \leq M'(p).$$

Liveness Problem

A net N(either a P/T net or a TAPN) is live if for every transition t of N and every reachable marking M, some marking in $[M\rangle$ enables t. The *liveness problem* for a net is the problem of deciding if it is live.

Deadlock Problem

A marking of a net is a *deadlock* if it enables no transitions. The *deadlock problem* for a net N(either a P/T net or a TAPN) is the problem of deciding if any of its reachable markings is a deadlock.

Selection of Results

For Place/Transition nets, it is known that the deadlock and reachability problems are polynomialy equivalent and deadlock problem is polynomial time reducible to the liveness problem [9, 12, 18], and that they are all decidable and

EXPSPACE-hard [20, 26, 11].

Similarly, in the 1-safe case, the three problems are all PSPACE-complete [11].

For timed-arc Petri nets, the reachability problem is undecidable [15], while the coverability problem is decidable [4]. Very recent results show that deadlock and liveness are also undecidable [29].

2.1.4 Verification Tools for Petri Nets

Nowadays we have several verification tools for Petri nets, such as Roméo [32] and Tina [36]. Roméo is used for time Petri nets analysis, and provides several methods for translating TPNs to TA and computation of state class graphs. It is developed by the Real-Time Systems team in the Communications and Cybernetic Research Institute of Nantes.

To the best of our knowledge, there is no verification tool for timed-arc Petri nets. To investigate verification approaches to timed-arc Petri nets is the aim of our project.

2.2 Timed Automata

2.2.1 Description

Timed automata [1] were first introduced by Alur and Dill in 1990 and has since then been established itself as a standard model for real-time systems.

Timed automata are finite state automata extended with a number of real valued clocks. Graphically a timed automaton can be depicted as nodes with arrows going from one node to another when there is a transition. We write *constraints* (also known as *guards*) at the origin of a transition and reset sets at the destination of the transition. At the center of the arrow we write the label.

In Figure 2.9 we have a very simple automaton with only two states and one transition. The transition goes from the initial state S0 to the state S1. The initial state is marked with double circles. The guard consists of only one atomic formula saying that the value of clock x should be less than 3. Similarly only one clock is reset (`x:=0`). The label on the transition is `'c!'`. This is the complement action of `'c?'`, which means that this transition must synchronize with an `'c?'` transition in another timed automaton. As in CCS [27] we can also have transitions with no label, these transitions are in fact τ transitions that does not need to synchronize. Figure 2.9 illustration was made using the graphical interface for Uppaal (see Section 2.2.5).

2.2.2 Timed Automata

Following are some auxiliary definitions related to timed automata models:

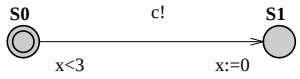


Figure 2.9: A Simple Timed Automaton in UPPAAL

Actions Let *Chan* be a finite set of channels, ranged over by c. We define Act to be a finite set of actions ranged over by a. For each channel in *Chan* we define two actions such that $Act = \{c!|c \in Chan\} \bigcup \{c?|c \in Chan\}$. We define a complement operator $^-$: $Act \rightarrow Act$ as $\bar{c}! = c?$ and $\bar{c}? = c!$. We define Δ to represent an infinite set of delay actions,

$$\Delta = \{\epsilon(d)|d \in \mathbb{R}^+\}$$

where we use $\mathbb{R}^+$ to stand for the non-negative reals. The special internal action is represented by τ. We define the two sets

$$Act_\tau = Act \cup \{\tau\}$$

$$\Delta_\tau = \Delta \cup \{\tau\}.$$

Clocks and Constraints Let C is a finite set of real valued clocks ranged over by x, y, z. A clock valuation

$$u : C \rightarrow \mathbb{R}^+$$

is a function that assigns to each clock a real non-negative value. We also define

$$\mathbb{R}^{+^C}$$

to be set of all clock valuations. We write $u(x)$ to mean the value of the clock x in the clock valuation u. We define two operations on clock valuations: `Reset` and `Delay`.

In `Reset` a set of clocks is set to zero:

$$u' = u[r \mapsto 0],\ r \subseteq C$$

defined by

$$\forall x \in r.u'(x) = 0 \text{ and } \forall x \in C \backslash r.u'(x) = u(x)$$

In `Delay` all clocks are increased with the same value:

$$u + d : C \rightarrow \mathbb{R}^+ \text{ where } d \in \mathbb{R}^+,$$

defined by

$$\forall x \in C.(u + d)(x) = u(x) + d.$$

We define $\mathcal{B}$(C) to be the set of all clock *constraints* (also known as *guards*), $g ::= A \mid g \wedge g$ where A is an atomic formula of the form: $x \prec n$ or $x - y \prec n$ for $\prec \in \{=, \leq, \geq, <, >\}$ and n being a natural number. We write $g(u) \models \texttt{tt}$ to mean that the clock constraint g is true under clock valuation u.

Then we can formally define a timed automaton which is basically a finite automaton extended with real-valued clocks as follows:

Definition 13. *(Timed Automata) A timed automaton is defined by a tuple* $A = (Act_\tau, L, l_0, C, E)$ *where*

1. Act_τ *is a set of actions defined above,*
2. L *is a finite set of locations (or nodes),*
3. $l_0 \in L$ *is the initial location,*
4. C *is a finite set of clocks,*
5. $E \subseteq L \times \mathcal{B}(C) \times Act_\tau \times 2^C \times L$ *is a set of edges. The tuple* $e = (l, g, \mu, r, l') \in E$ *stands for an edge from location* l *to location* l' *(the* target *of* e*) with action* μ*, where* r *denotes the set of clocks to be reset to 0 and* g *is the enabling condition (or* guard*) over the clocks of* A*.*

The semantics of timed automata is described in terms of a labelled transition system, defined by the triple $(S, \mathcal{L}, \rightarrow)$*, such that*

1. $S = L \times \mathbb{R}^{+C}$*, i.e., the states (***configurations***) consist of a node (location) and a clock valuation,*
2. $\mathcal{L} = Act_\tau \cup \Delta$*, i.e., the labels are the union of actions and delay actions, and*
3. $\rightarrow$ *is the transition relation which is defined as:*

$$(l, u) \xrightarrow{a} (l', u') \text{ iff } \exists (l, g, a, r, l',) \in E.\ g(u) \models \texttt{tt} \wedge u' = [r \mapsto 0]u$$
$$(l, u) \xrightarrow{\epsilon(d)} (l', u') \text{ iff } \exists d \in \mathbb{R}^+.\ l' = l \wedge u' = u + d.$$

A configuration of a timed automaton A *is a pair* $(l, u) \in L \times \mathbb{R}^{+C}$*. The initial configuration a TA* A *is defined as a pair* (l_0, u_0) *where*

1. l_0 *is the initial location, and*
2. u_0 *is the zero-valuation where* $\forall x \in C. u_0(x) = 0$*.*

In a TA A, if there is a sequence of configurations $(l_0, u_0), (l_1, u_1), ..., (l_m, u_m)$, such that

$$(l_0, u_0) \xrightarrow{a_1} (l_1, u_1) \xrightarrow{a_2} \cdots \xrightarrow{a_m} (l_m, u_m)$$

where

$$a_i \in Act_\tau \cup \Delta,\ (i = 1, 2, ...m),$$

then we say that the configuration (l_m, u_m) is **reachable** from (l_0, u_0), and denote this by

$$(l_0, u_0) \rightarrow^* (l_m, u_m).$$

2.2.3 Network of Timed Automata

Definition 14. *(Timed Automata Networks) A network of timed automata (short in TAN) N over actions Act_τ and clocks C has the form:*

$$N = A_1|A_2|A_3...|A_n$$

where each A_i (called process*) is a timed automaton over actions Act and clocks C.*

The clocks are all potentially global, but may in reality be local by being used in only one automaton. In the following definition of the semantics, we write $\overrightarrow{l}$ to mean a vector $(l_1, l_2, l_3, ..., l_n)$ of locations in each automaton A_i.

Definition 15. *(Updating Rule of TAN) A TAN $N = A_1|A_2|A_3...|A_n$ over actions Act and clocks C defines a labelled transition system $(S, \mathcal{L}, \rightarrow)$ where each of the states is a location (or node) in each timed automaton A_i and a clock valuation $S = L_1 \times L_2 \times \cdots \times L_n \times \mathbb{R}^{+C}$, the labels are $\mathcal{L} = \Delta_\tau$, and the transition relation $\rightarrow$ is defined as:*

- $(\overrightarrow{l}, u) \xrightarrow{\tau} (\overrightarrow{l'}, u')$ *iff for some* $i, j \in \{1, ..., n\}$ *where* $i \neq j$ *and* $a \in Act$,
  $\exists(l_i, g_i, a, r_i, l_i') \in E_i$
  $\exists(l_j, g_j, \bar{a}, r_j, l_j') \in E_j$
  $g_i(u) \wedge g_j(u) \models \texttt{tt},\ u' = [r_i \cup r_j \mapsto 0]u$
  $\forall k \notin \{i, j\}.l_k' = l_k$.
- $(\overrightarrow{l}, u) \xrightarrow{\tau} (\overrightarrow{l'}, u')$ *iff for some* $i \in \{1, ..., n\}$,
  $\exists(l_i, g_i, \tau, r_i, l_i') \in E_i$
  $g_i(u) \models \texttt{tt}, u' = [r_i \mapsto 0]u$
  $\forall k \notin \{i\}.l_k' = l_k$.
- $(\overrightarrow{l}, u) \xrightarrow{\epsilon(d)} (\overrightarrow{l'}, u')$ *iff* $\overrightarrow{l'} = \overrightarrow{l}$ *and* $u' = u + d$ *for all* $d \in \mathbb{R}^+$.

The three types of transitions presented above can be described respectively as *synchronizing*, *private*, and *delay* transitions. The first is synchronizing because two timed automata synchronize by taking transitions labelled with each others complement. The second is private because it involves only one timed automaton. The third is a delay transition where all clocks are increased by the same value.

Synchronous communication between the processes is by hand shake synchronization using input and output actions. Following Figure 2.10 is an example to illustrate the semantics. We have two simple timed automata that we combine into the TAN $N = S \mid T$.

2.2.4 Selection of Problems in TA

Reachability Problem

The *reachability problem* for a TA A with initial configuration (l_0, u_0) is the problem of deciding for a given configuration (l, u) of A if it is reachable from

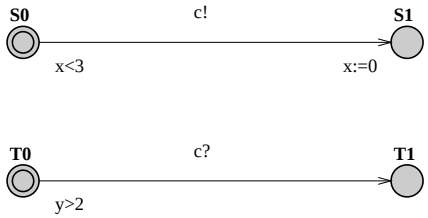


Figure 2.10: An example of TAN consists of two simple TA S and T

the configuration (l_0, u_0), i.e., to decide if

$$(l_0, u_0) \rightarrow^* (l, u).$$

Deadlock Problem

A configuration (l, u) of a TA is a *deadlock* if $(l, u) \not\rightarrow$, i.e., it cannot reach another configuration. The *deadlock problem* for a TA A with initial configuration (l_0, u_0) is the problem of deciding if any of its reachable configuration is a deadlock.

Results

These problems are known to be PSPACE-complete [2].

2.2.5 Verification Tool: Uppaal

This part is taken from [17]. UPPAAL [6] is a tool for modeling, simulation and verification of real-time systems. UPPAAL is being developed jointly by Uppsala University and Aalborg University. UPPAAL usage is appropriate for systems that can be modeled as a collection of non-deterministic processes with finite control structure and real valued clocks, communicating through channels or shared variables. The verification is done by automatic model-checker engine. Using UPPAAL we can construct abstract models of many system, simulate its dynamic behavior, specify and verify its safety properties and bounded liveness properties which can be useful to analyze and design embedded systems and real time systems. It consists of two main parts: a graphical user interface and a model-checker engine. It requires that Java 1.4 or higher is installed on the computer (for the latest version UPPAAL 3.5.2, Java 1.5 is needed). The engine part is by default executed on the same computer as the user interface, but can also run on a more powerful server.

UPPAAL has three main utilities: a *description language*, a *simulator*, and a *model-checker*. The description language is useful to represent the system as

collection of inter-related Timed Automata. Description language also has several data variables which can be applied on these Timed Automata designed with its help. The simulator and verifier are useful to graphically analyze the system behavior by considering various constraints on state space generated by the system. The simulator graphically represents the system behavior, it represents each and every transition the system took and the corresponding values for each transition. Verifier on the other hand is useful to check the properties of the system. In the query box we can verify several logic properties whether its satisfiable or not for the whole state space generated by the system.

UPPAAL is based on timed automata, that is finite state machine with clocks. The clocks are the way to handle time in UPPAAL. Time is continuous and the clocks measure time progress. It is allowed to test the value of a clock or to reset it. Time will progress globally at the same pace for the whole system. A system in UPPAAL is composed of concurrent processes, each of them modeled as an automaton. The automaton has a set of locations. Transitions are used to change location. To control when to fire a transition, it is possible to have a guard and a synchronization. A guard is a condition on the variables and the clocks saying when the transition is enabled. The synchronization mechanism in UPPAAL is a hand-shaking synchronization: two processes take a transition at the same time, one will have a `a!` and the other a `a?`, a being the synchronization channel. When taking a transition actions are possible: assignment of variables or reset of clocks. The following example which is illustrated in Figure 2.11 will make you familiar with this short description.

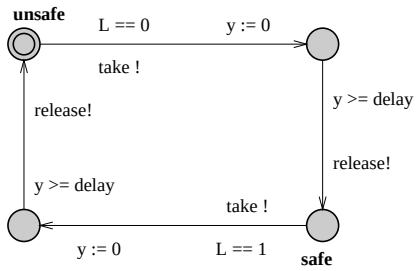


Figure 2.11: Example of a timed automaton in UPPAAL

3 Translations from TAPN to TA

This chapter deals with the reduction from TAPN into TA. First we will present basic idea of the reduction techniques/algorithms and then give the construction in a formal way.

3.1 Basic Idea of Translation

In this report, we focus on the translation of k-conservative TAPN only. Intuitively, unbounded number of tokens will lead to unbounded number of templates in our reduction techniques. More precisely, a variant number of tokens will lead to a variant number of templates. Thus we discuss only conservative TAPN in the report. In this chapter, otherwise is stated, all the TAPN mentioned below means a conservative one.

We study several cases of k-conservative TAPN and notice that in many conservative TAPN, some of its transitions will never be fired only because of the syntactical reasons. Which means that once such a transition is fired, the number of the tokens will not be kept as k. Since these transitions will never be fired, we call them **useless transitions**.

Thus in general, we follow two steps to implement the translation reduction from a TAPN to a TA. First, a TAPN N is reduced to a new TAPN N' which has the same properties while has no useless transitions. In the second step, we translate N' into the corresponding timed automaton TA. That means that the whole reduction can be shown as:

$$N \rightarrow N', \text{ and then } N' \rightarrow TA.$$

3.2 Removing Useless Transitions

As a start, we begin with the 1-conservative TAPN, and then go into the k-conservative TAPN.

A transition t in a Petri net can be fired only if each of its input places has at least one token which has qualified age. And once t is fired, each of its output arcs adds a token to each postcondition place. That means in a 1-conservative TAPN, no transitions with two or more input arcs can be fired, as there cannot be two or more places holding at least one token each at the same time. Nor can a transition with two or more output arcs be fired. In this case the net would generate more than one token after firing, which is forbidden in 1-conservative nets. Thus we can remove all the *useless* transitions in the first step and define them as those transitions which have either more than one input place or more than one output place. The set of *useful transitions* and *useless transitions* in a 1-conservative net are

$$T_{useful} \stackrel{def}{=} \{t \in T \mid |{}^\bullet t| = |t^\bullet| = 1\},$$

$$T_{useless} \stackrel{def}{=} T - T_{useful}.$$

Then, let us have a look at a more complex class, called 2-conservative nets. These are Petri nets in which every reachable marking always has exactly 2 tokens. Because neither of the two tokens can be consumed, there cannot be a transition where $|{}^\bullet t| > |t^\bullet|$ is enabled. Since no tokens could be generated as well, transitions where $|{}^\bullet t| < |t^\bullet|$ will not be enabled, either. We define the *useful transitions* where

$$T_{useful} \stackrel{def}{=} \{t \in T \mid 1 \leq |{}^\bullet t| = |t^\bullet| \leq 2\}.$$

And then, we have

$$T_{useless} \stackrel{def}{=} T - T_{useful}.$$

With a further research based on the 1-conservative nets and 2-conservative nets, we discover several qualities of the transitions of k-conservative nets. A *useful transition* of a k-conservative net could be defined in the following way.

Definition 16. *The useful transitions of a k-conservative net N is a set of transitions which have the same number of input arc(s) and output arc(s) no greater than k, formally defined as:*

$$T_{useful} \stackrel{def}{=} \{t \in T | 1 \leqslant |{}^\bullet t| = |t^\bullet| \leqslant k\}$$

$$T_{useless} \stackrel{def}{=} T - T_{useful}.$$

Let N be a k-conservative TAPN $(P, T, F, \texttt{times})$, and N' be the modified one $(P', T', F', \texttt{times}')$ which has been removed all *useless* transitions as follows,

1. $P' \stackrel{def}{=} P$,
2. $T' \stackrel{def}{=} T - T_{useless}$,
3. $F' \stackrel{def}{=} \{(p,t) \in F |\ t \in T'\} \cup \{(t,p) \in F |\ t \in T'\}$,
4. $\texttt{times}'(p,t) = \texttt{times}(p,t)$ where $p \in P'$, $t \in T'$.

Theorem 1. *A marking M is reachable in N from the initial marking M_0 iff M is reachable in N' from the initial marking M_0, i.e.,*

$$M \in [M_0\rangle \text{ in } N \Leftrightarrow M \in [M_0\rangle \text{ in } N'.$$

Proof.

- " $\Rightarrow$ ": Assume $M \in [M_0\rangle$ in N and M is reached from M_0 by an occurrence sequence $\sigma = t_1, t_2, t_3, ...t_n$. Because each transition t_i in σ $(i = 1, 2, ...n)$ is fired, it means that $t_i \in T_{useful}$ and t_i will not be removed as a useless transition. Such that we can find the corresponding t_i in N' which means that σ can be fired also in N'. Since the initial markings in N and N' are equal, we have $M_0[\sigma\rangle M$ in N' which means

$$M \in [M_0\rangle \text{ in } N'.$$

- " $\Leftarrow$ ": Let us consider the opposite direction. Assume $M \in [M_0\rangle$ in N' and M is reached from M_0 by an occurrence sequence $\sigma = t_1, t_2, t_3, ...t_n$. Because for every transition t_i $(i = 1, 2, ...n)$, it is true that $t_i \in T'$ and $T' \stackrel{def}{=} T - T_{useless}$, then $t'_i \in T$, and hence σ is an occurrence sequence also in N. Therefore, we have $M_0[\sigma\rangle M$ in N, which implies

$$M \in [M_0\rangle \text{ in } N.$$

□

Here is an example of removing useless transitions in a 2-conservative TAPN. In Figure 3.1 we illustrate an original TAPN, and we see how the useless transitions are removed.

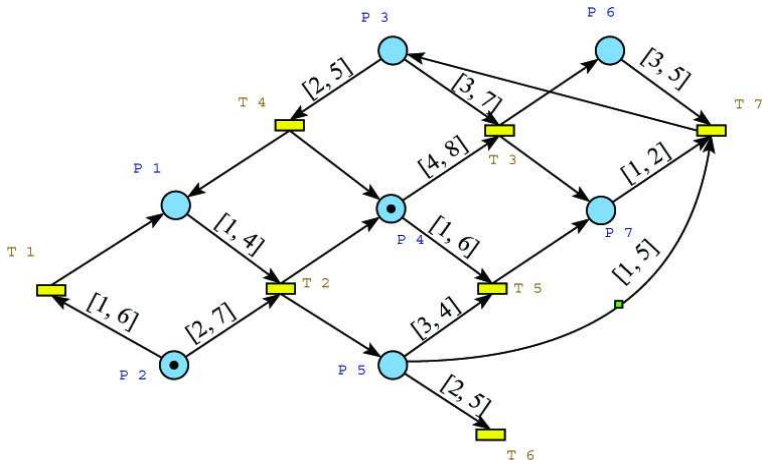


Figure 3.1: Original 2-Conservative TAPN Contains Useless Transitions

According to the removing rules, we remove the transitions T_4, T_5, T_6, and T_7, and then get the modified TAPN (see Figure 3.2).

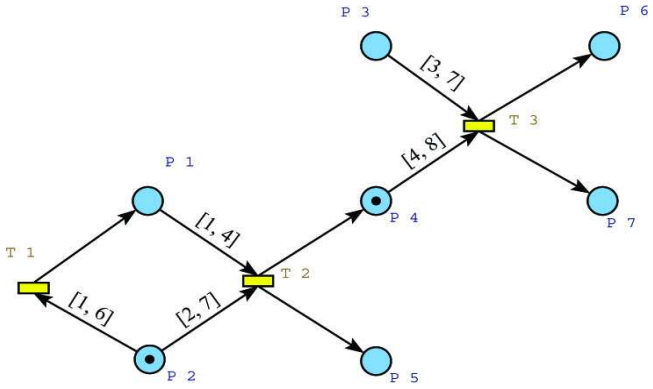


Figure 3.2: Modified TAPN Contains No Useless Transitions

Remark 1. *Unless otherwise stated, all TAPNs mentioned afterwards in this chapter are assumed to be those without useless transitions.*

After we got a TAPN that contains no useless transitions, we then are able to translate it into TA/TAN.

3.3 Reduction of Degree of TAPN

First, we introduce a terminology of *degree* in k-conservative TAPN.

Definition 17. *Let N be a k-conservative TAPN which contains only useful transitions. As before, we can write*

$$T_{useful} \stackrel{def}{=} T_1 \cup T_2 \cup \cdots \cup T_k$$

where

$$T_i = \{t \mid |{}^\bullet t| = |t^\bullet| = i\} \ (i = 1, 2, \cdots k).$$

We say that N is of **degree** *d ($d \in \{0, 1, \cdots, k\}$) iff $T_i = \emptyset$ for all i where $d < i \leq k$.*

Notice the different synchronization mechanisms between the firing rule of a TAPN and what we can simulate in UPPAAL, the maximum number of synchronization channels limits us that we cannot reduce TAPN with a degree higher than 2, directly into a TAN.

So before translating, we have to modify the TAPN to make it suitable to be translated. The idea is to reduce the degree of a k-conservative TAPN step by step, until it is with a degree no more than 2, then we are able to reduce it into TA/TAN using the synchronization mechanism in UPPAAL.

3.3.1 Reduction from 3-Conservative TAPN to 3-Conservative TAPN of Degree 2

To give an easy beginning, we start at the reduction from 3-Conservative TAPN to 3-Conservative TAPN of Degree 2.

Let $N = (P, T, F, \texttt{times})$ be a 3-conservative TAPN, we have $T_{useful} = T_1 \cup T_2 \cup T_3$.

Definition 18. *Given a 3-conservative TAPN $N = (P, T, F, \texttt{times})$, we define a reduced 3-conservative TAPN $N' = (P', T', F', \texttt{times}')$ of degree 2 in the following way:*

1. $P' \stackrel{def}{=} P \cup \{p_t^1, p_t^2, p_t^3 \mid t \in T_3\}$;
2. $T' \stackrel{def}{=} T_1 \cup T_2 \cup \{t^1, t^2, t^3 \mid t \in T_3\}$;
3. (a) *for all $t \in T_1 \cup T_2$,*
   - $F_1 \stackrel{def}{=} \{(p,t), (t,p') \mid t \in T_1 \cup T_2 \text{ and } p \in {}^\bullet t,\ p' \in t^\bullet\}$, *and*
   - $\texttt{times}'(p,t) \stackrel{def}{=} \texttt{times}(p,t)$ *where $p \in {}^\bullet t$,*

   (b) *for any $t \in T_3$, let us fix $\{p_1^t, p_2^t, p_3^t\} = {}^\bullet t$ and $\{q_1^t, q_2^t, q_3^t\} = t^\bullet$, then t is replaced by t^1, t^2, t^3 in the following way:*
   - $F_2 \stackrel{def}{=} \{(p_1^t, t^1), (p_2^t, t^1), (t^1, p_t^1), (t^1, p_t^2), (p_t^2, t^2), (p_3^t, t^2), (t^2, p_t^3), (t^2, q_3^t), (p_t^1, t^3), (p_t^3, t^3), (t^3, q_1^t), (t^3, q_2^t) \mid t \in T_3\}$, *and*
   - *the function $\texttt{times}'$ is defined as:*
     - $\texttt{times}'(p_1^t, t^1) \stackrel{def}{=} \texttt{times}(p_1^t, t)$
     - $\texttt{times}'(p_2^t, t^1) \stackrel{def}{=} \texttt{times}(p_2^t, t)$
     - $\texttt{times}'(p_3^t, t^2) \stackrel{def}{=} \texttt{times}(p_3^t, t)$
     - $\texttt{times}'(p_t^1, t^3) \stackrel{def}{=} [\ 0, 0\ ]$
     - $\texttt{times}'(p_t^2, t^2) \stackrel{def}{=} [\ 0, 0\ ]$
     - $\texttt{times}'(p_t^3, t^3) \stackrel{def}{=} [\ 0, 0\ ]$.

   $F' = F_1 \cup F_2$.

Figure 3.3 and Figure 3.4 demonstrate a simple example of the reduction.

For the further study on the reduction, we introduce a function f to present the correspondence between N and N'.

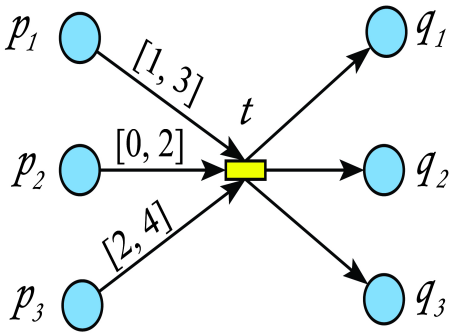


Figure 3.3: A Given TAPN with degree 3

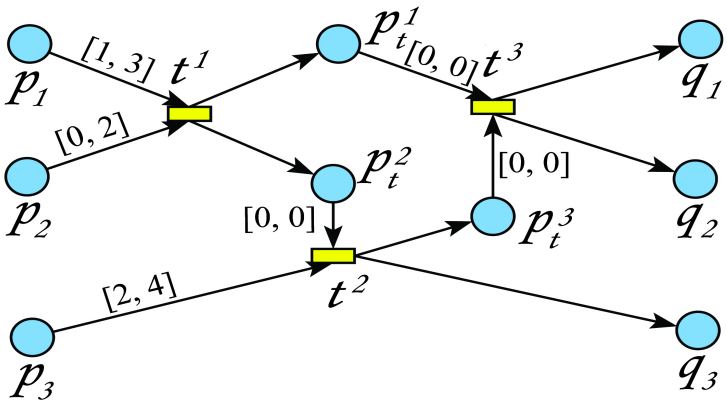


Figure 3.4: The Reduction from 3-Degree to 2-Degree

Definition 19. *(Correspondence Function) For a marking M in a given TAPN N and the degree-reduced TAPN N', we define $f(M)$ in N',*

$$f: \ (P \to \mathcal{B}(\mathbb{R})) \longrightarrow (P' \to \mathcal{B}(\mathbb{R}))$$

such that

$$f(M(p)) = \begin{cases} M(p) & p \in P \\ \emptyset & p \notin P. \end{cases}$$

In the meanwhile, we introduce some terminology.

Definition 20. *Given a 3-degree TAPN N and the degree-reduced TAPN N', we say that a marking M' in N' is* **stable** *iff*

$$\exists M \text{ in } N \text{ s.t. } f(M) = M'.$$

Any other M' in N' which is not stable, we call an **intermediate** *marking. We say that a place $p \in P'$ is* **original** *if $p \in P$, otherwise we call it* **intermediate**. *Similarly, we say that a transition $t \in T'$ is* **original** *if $t \in T$, otherwise we call it* **intermediate**.

Recall the example in Figure 3.4, we can see that $p_1, p_2, p_3, q_1, q_2, q_3$ are original places while p_t^1, p_t^2, p_t^3 are intermediate places.

Theorem 2. *A 3-conservative TAPN N is polynomial time reducible to a 3-conservative TAPN N' with degree 2, preserving reachability, i.e., for a marking M of a 3-conservative TAPN N with the initial marking M_0, we have*

$$M \in [M_0\rangle \text{ in } N \text{ iff } f(M) \in [f(M_0)\rangle \text{ in } N'. \tag{3.1}$$

Proof. We will show the proof for a single transition firing, which naturally generalizes to a sequence of transitions.

1. $M_0[t\rangle M \Rightarrow f(M) \in [f(M_0)\rangle$
   Assume transition t is fired in this step, so there are two situations as follows:

   (a) $t \in T_1 \cup T_2$
   According to Definition 19 and Definition 18, it is easy to see that

   $$f(M_0)[t\rangle f(M).$$

   (b) $t \in T_3$
   According to Definition 18, we can see that three intermediate transitions t^1, t^2, t^3 and three intermediate places p_t^1, p_t^2, p_t^3 are generated. Hence there are intermediate markings M' and M'' in N' s.t.

   $$f(M_0)[t^1\rangle M'[t^2\rangle M''[t^3\rangle f(M)$$

   according to Definition 19.

2. $f(M) \in [f(M_0)\rangle \Rightarrow M \in [M_0\rangle$
   For any stable marking M' in N', we consider only two cases:

   - M' is reachable from another stable marking M_0' in 1 step.
   - M' is reachable from another stable marking M_0' through some intermediate markings.

   (a) Without intermediate markings: $M_0'[t'\rangle M'$.
   According to Definition 19 and Definition 18, we know $t' \in T_1 \cup T_2$ and according to Definition 20, there are M_0, M in N s.t.

   $$f(M_0) = M_0' \text{ and } f(M) = M'.$$

   Now we can see that also $M_0[t'\rangle M$.

   (b) With intermediate markings, there are two cases as well.

- Through three intermediate transitions only;
- Through three intermediate transitions and some original transitions in T_1 or T_2.

i. Through only three intermediate transitions t^1, t^2, t^3 for some $t \in T_3$:
According to Definition 18, let us assume

$$M_1'[t^1\rangle M_2'[t^2\rangle M_3'[t^3\rangle M_4'$$

where M_1' and M_4' are stable markings, and M_2' and M_3' are intermediate ones.
Since the sequence of transitions t_1, t_2, t_3 are reduced from a $t \in T_3$, hence we know $t' \in T_3$ and according to Definition 20, there are M_1, M_4 in N s.t.

$$f(M_1) = M_1' \text{ and } f(M_4) = M_4'.$$

Hence we can see that $M_1[t\rangle M_4$.

ii. In the case that intermediate and original transitions are mixed, observe that no other transitions with two input places for some different t' can be fired, i.e., no transitions from T_2 and no intermediate transitions t^1, t^2, t^3 for $t' \neq t$ can be fired. In this case, we can see that besides t^1, t^2, t^3, there are transitions $t_1, \cdots, t_j \in T_1$. Thus we have

$$M_0' \xrightarrow{t^1} M_1' \xrightarrow{t_1} M_2' \cdots \xrightarrow{t_i} M_{i+1}' \xrightarrow{t^2} M_{i+2}' \xrightarrow{t_{i+1}} \cdots \xrightarrow{t_j} M_{j+2}' \xrightarrow{t^3} M'$$

where t^1, t^2, t^3 are the intermediate transitions and $t_k \in T_1$ $(k = 1, 2, ..., j)$, which is the most general case how such firing can look. According to the time constraints [0, 0] in the inputs of intermediate transitions, the sequence of transitions

$$t^1, t_1, \cdots t_i, t^2, t_{i+1}, \cdots t_j, t^3 \tag{3.2}$$

are fired without time elapsing, i.e., the transitions sequence 3.2 has the same effect as

$$t_1, t_2, \cdots, t_i, t^1, t^2, t^3, t_{i+1}, \cdots, t_j. \tag{3.3}$$

Since the corresponding relationship through sequence of transitions of $t \in T_1 \cup T_2$ has been proved before, and the sequence of transitions of t^1, t^2, t^3 is proved as well, according to Definition 20, we know there are $t' \in T_3$ and a sequence of transitions $t_1, t_2, \cdots, t_j \in T_1 \cup T_2$ and there are M_0, M in N s.t.

$$f(M_0) = M_0' \text{ and } f(M) = M'$$

and through a sequence of transitions, $M \in [M_0\rangle$.

□

Complexity Analysis

Let $N = (P, T, F, \texttt{times})$ be a general 3-conservative TAPN and $N' = (P', T', F', \texttt{times'})$ be the reduced 3-conservative TAPN with degree 2. Comparing to N, we have the space complexity of N' as below:

- $|P'| = |P| + 3|T_3|$
- $|T'| = |T_1| + |T_2| + 3|T_3|$
- $|F'| = |F| + 6|T_3|$

Theorem 3. *Let $N = (P, T, F, \texttt{times})$ be a general 3-conservative TAPN and $N' = (P', T', F', \texttt{times}')$ be the reduced 3-conservative TAPN with degree 2. The complexity of N' is linear in size of N.*

Proof.

$$
\begin{aligned}
&|N'| = |P'| + |T'| + |F'| \\
&= (|P| + 3|T_3|) + (|T_1| + |T_2| + 3|T_3|) + (|F| + 6|T_3|) \\
&= |P| + |T_1| + |T_2| + 12|T_3| + |F| \\
&\leqslant |P| + 12|T| + |F| \\
&\leqslant 12|N|.
\end{aligned}
$$

□

With further study, we find that the reduction of degree is safe even if there are more tokens in the net and it is k-conservative ($k > 2$), but still of degree 2.

Remark 2. *Any k-conservative TAPN of degree 3, it can be reduced to a k-conservative TAPN of degree 2, preserving the reachability property.*

3.3.2 Reduction from Conservative TAPN of Degree k to Degree k-1

Based on the study on the reduction of degree 3, we can see that the reduced conservative TAPN of degree 2 keeps the reachability property even if there are more tokens in the original TAPN of degree 3, which inspires us presenting the reduction from a conservative TAPN of degree k to degree k-1 where $k > 2$. Thus step by step, we finally can have a TAPN of only degree 2, which will be more suitable to be reduced into TAN.

Let $N = (P, T, F, \texttt{times})$ be a conservative TAPN of degree k, we have $T_{useful} = T_1 \cup T_2 \cup \cdots \cup T_k$.

Definition 21. *Given a conservative TAPN $N = (P, T, F, \texttt{times})$ of degree k, we define a reduced conservative TAPN $N' = (P', T', F', \texttt{times}')$ of degree k-1 in the following way:*

1. $P' \stackrel{def}{=} P \cup \{p_t^1, p_t^2, \cdots, p_t^{2k-3} \mid t \in T_k\}$;

2. $T' \stackrel{def}{=} T_1 \cup T_2 \cup \cdots \cup T_{k-1} \cup \{t^1, t^2, \cdots, t^k \mid t \in T_k\}$;

3. (a) *for all* $t \in T_1 \cup T_2 \cup \cdots \cup T_{k-1}$,

   - $F_1 \stackrel{def}{=} \{(p,t),(t,p') \mid p \in {}^\bullet t,\ p' \in t^\bullet\}$, *and*
   - $\texttt{times}'(p,t) \stackrel{def}{=} \texttt{times}(p,t)$ *where* $p \in {}^\bullet t$,

   (b) *for any* $t \in T_k$, *let us fix* $\{p_1^t, p_2^t, \cdots, p_k^t\} = {}^\bullet t$ *and* $\{q_1^t, q_2^t, \cdots, q_k^t\} = t^\bullet$, *then* t *is replaced by* $t^1, t^2, \cdots, t^k$ *in the following way:*

   - $F_2 \stackrel{def}{=} \{(p_i^t, t^1), (t^1, p_t^i), (p_t^{k-3+n}, t^n), (p_k^t, t^2), (p_t^{k+1-n}, t^n),$ $(t^k, q_1^t), (t^n, p_t^{k-2+n}), (t^n, q_{k+2-n}^t) \mid t \in T_k,$ $(i = 1, \cdots (k-1)),\ (n = 2, \cdots k)\}$, *and*
   - *the function* $\texttt{times}'$ *is defined as:*
     - $\texttt{times}'(p_i^t, t^1) \stackrel{def}{=} \texttt{times}(p_i^t, t)$
     - $\texttt{times}'(p_k^t, t^2) \stackrel{def}{=} \texttt{times}(p_k^t, t)$
     - *all other added arcs have the interval* $[\ 0, 0\ ]$

   where $i = 1, \cdots (k-1))$

$F' = F_1 \cup F_2$.

Figure 3.5 and Figure 3.6 demonstrate a simple example of the reduction from a TAPN with degree 5 to a TAPN with degree 4.

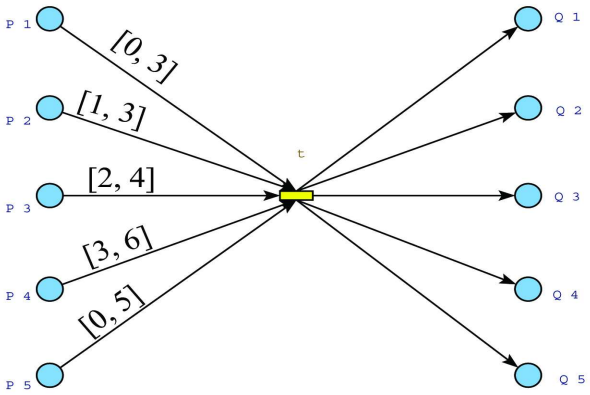


Figure 3.5: A Given TAPN with degree 5

Similar to Definition 19, we also have the correspondence function for a general conservative TAPN of degree k.

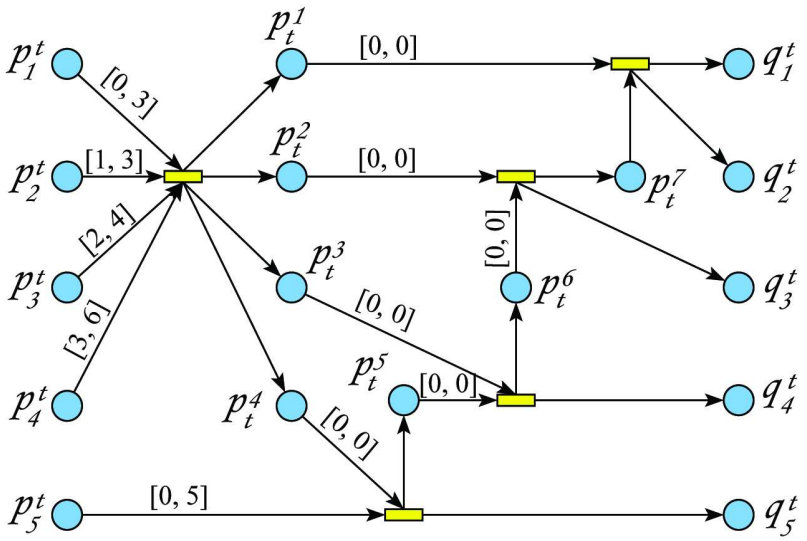


Figure 3.6: The Reduction from 5-Degree to 4-Degree

Definition 22. *(k-Degree Correspondence Function) For a marking M in a given TAPN N of degree k and the reduced TAPN N' of degree k-1, we define $f(M)$ in N',*

$$f: \ (P \to \mathcal{B}(\mathbb{R})) \longrightarrow (P' \to \mathcal{B}(\mathbb{R}))$$

such that

$$f(M(p)) = \begin{cases} M(p) & p \in P \\ \emptyset & p \notin P. \end{cases}$$

And as Definition 20, we modify the terminologies for a general conservative TAPN as follows:

Definition 23. *Given a k-degree TAPN N and the degree-reduced TAPN N', we say that a marking M' in N' is* **stable** *iff*

$$\exists M \text{ in } N \text{ s.t. } f(M) = M'.$$

Any other M' in N' which is not stable, we call an **intermediate** *marking. We say that a place $p \in P'$ is* **original** *if $p \in P$, otherwise we call it* **intermediate**. *Similarly, we say that a transition $t \in T'$ is* **original** *if $t \in T$, otherwise we call it* **intermediate**.

Theorem 4. *A conservative TAPN N of degree k is polynomial time reducible to a conservative TAPN N' with degree k-1 (k>2), preserving reachability, i.e., for a marking M of TAPN N with the initial marking M_0, we have*

$$M \in [M_0\rangle \text{ in } N \text{ iff } f(M) \in [f(M_0)\rangle \text{ in } N'. \tag{3.4}$$

Proof Sketch. The proof proceeds the same idea as the proof of Theorem 2. There are more situations that extra tokens and more complicate behaviors exist, but the basic idea lies on the interval [0, 0] limits all the tokens in intermediate places have to be fired instantly, otherwise will be dead. This mechanism ensures the correctness of the reduction.

Complexity Analysis

Let $N = (P, T, F, \texttt{times})$ be a general conservative TAPN of degree k and $N' = (P', T', F', \texttt{times'})$ be the reduced conservative TAPN degree k-1. Comparing to N, we have the space complexity of N' as below:

- $|P'| = |P| + (2k-3)|T_k|$
- $|T'| = |T_1| + |T_2| + \cdots + |T_{k-1}| + k|T_k|$
- $|F'| = |F| + (4k-6)|T_k|$

Remark 3. *Let $N = (P, T, F, \texttt{times})$ be a general conservative TAPN of degree k and $N' = (P', T', F', \texttt{times}')$ be the reduced conservative TAPN of degree k-1. N' has a polynomial size to N, i.e., the size($|N'|$) of N' is $O(k|N|)$.*

Proof.

$$\begin{aligned}
&|N'| = |P'| + |T'| + |F'| \\
&= (|P| + (2k-3)|T_k|) + (|T_1| + |T_2| + \cdots + |T_{k-1}| + k|T_k|) + (|F| + (4k-6)|T_k|) \\
&= |P| + |T_1| + |T_2| + \cdots + |T_{k-1}| + (7k-9)|T_k| + |F| \\
&\leqslant |P| + (7k-9)|T| + |F| \\
&\leqslant (7k-9)|N|.
\end{aligned}$$

Thus we have $|N'| = O(k|N|)$. □

In the meanwhile, the reader may also wonder why not to consider another reduction of degree k, as illustrated in Figure 3.7. For each single step, the complexity is:

- $|P'| = |P| + k|T_k|$
- $|T'| = |T_1| + |T_2| + \cdots + |T_{k-1}| + 2|T_k|$
- $|F'| = |F| + 2k|T_k|$.

It is easy to implement it a single step, but we construct the reduction in a recursive algorithm, generally. The problem of this reduction is that it generates an extra transition of degree k-1 for each k-degree transition in the original net, which makes the cost of the reduction exponentially increase. Thus we keep the first reduction, which has a polynomial cost in implementation.

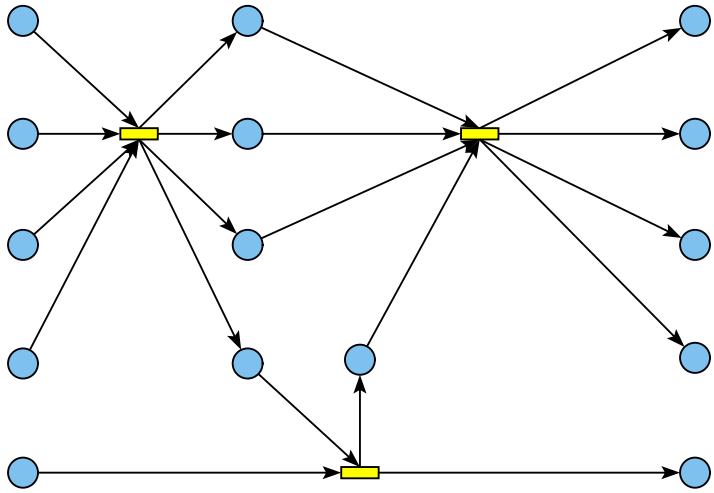


Figure 3.7: Another Reduction from 5-Degree to 4-Degree

3.4 Reduction from TAPN to TA/TAN

3.4.1 Reduction of 1-Conservative TAPN

When a TAPN is reduced as the degree is no more than 2, we are ready to translate into TA.

Among all the TAPN, 1-conservative case has the strongest constraint and could be the easiest one to start the discussion.

According to the removing rule, we can easily get the modified TAPN without useless transitions. We keep all the places in N as the states in TA, and all the useful transitions (with arcs) as the transitions in TA. The age of the token corresponds to a new clock x in TA. The place which has the initial token, is corresponding to the *initial* location in TA. Thus we can define the reduction from 1-conservative TAPN to timed automata in this formal way:

Definition 24. *(Reduction of 1-Conservative TAPN)*
Let $N = (P, T, F, \texttt{times})$ *with initial marking* M_0 *be a TAPN without useless transitions. We define the corresponding timed automaton as* $A(N) =$

(Act_τ, L, l_0, C, E) *such that*

1. $Act_\tau \stackrel{def}{=} \{\tau\}$,
2. $L \stackrel{def}{=} P$,
3. $l_0 \stackrel{def}{=} p$ *such that* $|M_0(p)| = 1$ *(this place is always unique because* M *is a 1-conservative marking),*
4. $C \stackrel{def}{=} \{x\}$,
5. E *is defined s.t. if* $t \in T$ *then* $(l, g, \tau, r, l') \in E$ *where*
   - $l \stackrel{def}{=} p$ *s.t.* $\{p\} = {}^\bullet t$,
   - $g \stackrel{def}{=} \pi_1(p,t) \leq x \wedge x \leq \pi_2(p,t)$ *where* $[\pi_1(p,t), \pi_2(p,t)] = \texttt{times}(p,t)$ *where* $\{p\} = {}^\bullet t$,
   - τ *is internal actions,*
   - $r \stackrel{def}{=} \{x\}$, *and*
   - $l' \stackrel{def}{=} p$ *s.t.* $\{p\} = t^\bullet$.

For the further study on the reduction of 1-conservative TAPN, we introduce a function f to present the correspondence relationship between TAPN and TA.

Definition 25. *(Correspondence Function) We define a correspondence function*

$$f : (P \to \mathcal{B}(\mathbb{R}^+)) \longrightarrow (L \times {\mathbb{R}^+}^C)$$

by

$$f(M) = (l, u),$$

where

1. $l \stackrel{def}{=} p$ *where* $|M(p)| = 1$, *and*
2. $u(x) \stackrel{def}{=} x_p$ *s.t.* $\{x_p\} = M(p)$.

Theorem 5. *1-conservative TAPNs are polynomial time reducible to timed automata, preserving reachability, i.e., for every reachable marking* M *of a 1-conservative TAPN* N*, we have*

$$M \in [M_0\rangle \text{ iff } f(M_0) \to^* f(M).$$

Proof. It is enough to prove that for every transition $t \in T$ where $M[t\rangle M'$ we have $f(M) \xrightarrow{\tau} f(M')$ and vice versa, and for every time increasing transition $M[\epsilon(d)\rangle M'$ we have $f(M) \xrightarrow{\epsilon(d)} f(M')$ and vice versa. Below, we give the proof

formally.

Let $N = (P, T, F, \texttt{times})$ with initial marking M_0 be a TAPN without useless transitions, and $A(N) = (Act_\tau, L, l_0, C, E)$ is the corresponding timed automaton as defined above.

1. " $\Rightarrow$ ":
   (a) $M[t\rangle M' \Rightarrow f(M) \xrightarrow{\tau} f(M')$

   Because M and M' are markings of N, according to Definition 25, we have the corresponding configurations (l, u) and (l', u'), respectively, such that,
   - $l = p$ where $|M(p)| = 1$, $u(x) = x_p$ s.t. $\{x_p\} = M(p)$, and
   - $l' = p'$ where $|M'(p')| = 1$, $u'(x) = x_{p'}$ s.t. $\{x_{p'}\} = M'(p')$.

   According to Definition 10, firing rules of TAPN, we have $\{p\} = {}^\bullet t$ and $\{p'\} = t^\bullet$.
   Then according to Definition 24, we have
   - $l = p$ where $\{p\} = {}^\bullet t$ and $l' = p'$ where $\{p'\} = t^\bullet$,
   - $u(x) = x_p$ s.t. $\{x_p\} = M(p)$ and $u'(x) = x_{p'}$ s.t. $x_{p'} = 0$,
   - $g(u) \models \texttt{tt}$.

   Hence we have $(l, u) \xrightarrow{\tau} (l', u')$ according to Definition 13.

   (b) $M[\epsilon(d)\rangle M' \Rightarrow f(M) \xrightarrow{\epsilon(d)} f(M')$

   Next, consider $M[\epsilon(d)\rangle M'$ $(d \in \mathbb{R}^+)$ s.t. $f(M) = (l, u)$ and $f(M') = (l', u')$.

   Then according to Definition 10, Definition 24 and Definition 25, we have
   - $l = l'$, and
   - $u(x) = x_p$ s.t. $\{x_p\} = M(p)$ and
     $u'(x) = x_{p'}$ s.t. $x_{p'} = x_p + d$, where $\{x_p\} = M(p)$.

   Hence we have $(l, u) \xrightarrow{\epsilon(d)} (l', u')$ according to Definition 13.

2. " $\Leftarrow$ "
   (a) $f(M) \xrightarrow{\tau} (l', u') \Rightarrow$ there is $t \in T$ s.t. $M[t\rangle M'$ and $f(M') = (l', u')$

   Let M be a marking and $f(M) = (l, u)$, such that $(l, u) \xrightarrow{\tau} (l', u')$. Due to $(l, g, \tau, \{x\}, l') \in E$, and according to Definition 24, there is $t \in T$ such that

- $l = p$ where $\{p\} = {}^\bullet t$,
- $g(u) = x \geq \pi_1(p,t) \wedge x \leq \pi_2(p,t) \models \texttt{tt}$,
- $l' = p$ where $\{p\} = t^\bullet$.

According to Definition 24, we have $u'(x) = 0$.
Thus let M' be a marking s.t. $M[t\rangle M'$ and $f(M') = (l_1, u_1)$.
Then according to the Definition 25, we have

- $l_1 = p$ where $\{p\} = t^\bullet$, and
- $u_1(x) = 0$.

According to Definition 24, we have $l_1 = l'$ and $u_1(x) = 0 = u'(x)$,
therefore $(l_1, u_1) = (l', u')$.
Hence $M[t\rangle M'$ such that $f(M') = (l', u')$.

(b) $f(M) \xrightarrow{\epsilon(d)} (l', u') \Rightarrow M[\epsilon(d)\rangle M'$ s.t. $f(M') = (l', u')$

Let M be a marking and $f(M) = (l, u)$, such that $(l, u) \xrightarrow{\epsilon(d)} (l', u')$.
First, according to Definition 13, we have
$l' = l$, and
$u'(x) = u(x) + d$.
Thus let M' be a marking where $M[\epsilon(d)\rangle M'$,
and $f(M') = (l_1, u_1)$.
Then according to Definition 25, we have

- $l_1 = p$ where $|M'(p)| = 1$, and
- $u_1(x) = x'_p$ s.t. $\{x'_p\} = M'(p)$ where $x'_p = x_p + d$, where $\{x_p\} = M(p)$.

Hence $l_1 = l = l'$, and $u_1(x) = u(x) + d = u'(x)$,
which means that $(l_1, u_1) = (l', u')$,
This gives $M[t\rangle M'$ such that $f(M') = (l', u')$.

From the above, we prove that the corresponding relation between TAPN markings and TA configurations is 1-to-1 mapping. Which means, for a sequence of markings in TAPN N, we have a corresponding sequence of configurations in TA A and the other way round.

□

Given a simple example of TAPN N, which has only the useful transitions T_1, T_2, T_3, as illustrated in Figure 3.8. According to the reduction rule, we get the corresponding timed automaton TA, as illustrated in Figure 3.9.

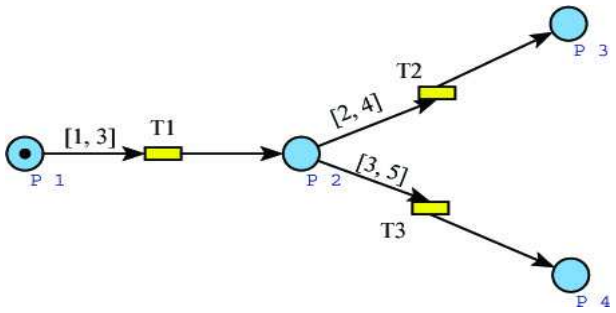


Figure 3.8: An Example of A Modified 1-Conservative TAPN

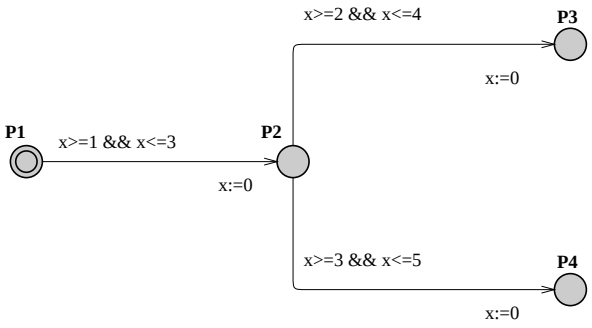


Figure 3.9: Translated 1-Conservative TAPN Model in Timed Automaton

Remark 4. *For a TAPN, if for all transitions we have*

$$|^\bullet t| = |t^\bullet| = 1,$$

but there are more tokens in the initial marking, then we can translate it into network of TA according to the same technique which is used in the reduction of 1-conservative TAPN, and the number of tokens will correspond to the number of parallel components.

3.4.2 Reduction of 2-Conservative TAPN

As we have the modified TAPN (see Figure 3.2) which has removed all the useless transitions, now we can go on with the reduction to timed automaton.

To illustrated the 2-conservative TAPN reduction algorithm we give an example. In this example we explain the construction of TA from a given 2-conservative TAPN.

First, recall what we have discussed in Section 3.2. The useful transitions in a 2-conservative TAPN are $T_{useful} = T_1 \cup T_2$ where

- $T_1 = \{t \in T \mid |^\bullet t| = |t^\bullet| = 1\}$, and
- $T_2 = \{t \in T \mid |^\bullet t| = |t^\bullet| = 2\}$.

For those $t \in T_1$, the construction is the same as the construction of TA from 1-conservative TAPN. The 2-conservative TAPN is allowed to have these kind of transitions as well.

While for those $t \in T_2$, the construction is altered at the places wherever such transitions exists.

Synchronization Problem and Solution

In a 2-conservative TAPN, the most different point other than 1-conservative TAPN is that there is one more kind of transition which has two input arcs and two output arcs, thus $|^\bullet t| = |t^\bullet| = 2$. The simple reduction model for 1-conservative TAPN is lacking the capability to make this reduction to timed automata. In other words, simply making a copy of all the places and all the useful transitions can not simulate the performance of the TAPN in timed automata.

At such transition places we use the communication channels present in timed automata. In TA, channels are used to synchronize the processes. This is done by annotating edges in the model with *synchronization labels*.

These communication channels allow the parallel composition of two different templates. Here one template will be communicating with the other template, in other words, one template will make calls while the other template listens to

the calls. The communication which is possible in TA and is handshake communication. The communication is achieved using the operators `!` and `?`.

Now let us take the 2-conservative TAPN (see Figure 3.2) as the example. Following is how we make the reduction. As in the example of 2-conservative TAPN we can see two kinds of transitions. We reduce in the following way.

We construct two templates (see Figure 3.11) with the same number of places as we have in the original 2-conservative TAPN.

The places P2 and P4 with tokens in the initial marking from Figure 3.2 were marked as initial places in each of the two TAs.

1. We reduce T1 in the TAPN into the transition in TA the same way as in the 1-conservative TAPN reduction.
2. For T2 and T3, each of them is replaced by transitions between places which have the outgoing transitions to the places which has input transitions. We mark each transition edge with a channel such that a transition channel is communicating with the transition channel in the other template transition.

As we can see in the figure for the transition T2 in 2-conservative TAPN we have two templates $P1$ and $P2$.

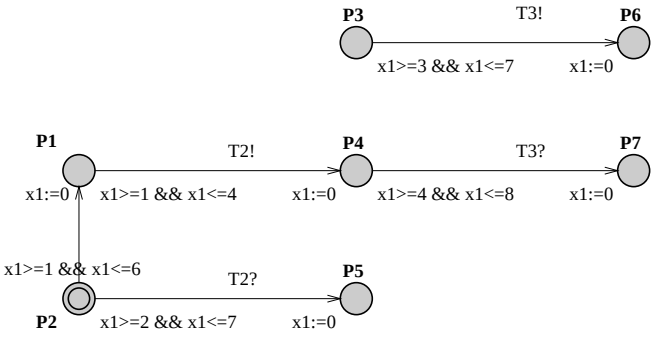


Figure 3.10: Template $P1$ in the Corresponding TAN Model for 2-Conservative TAPN

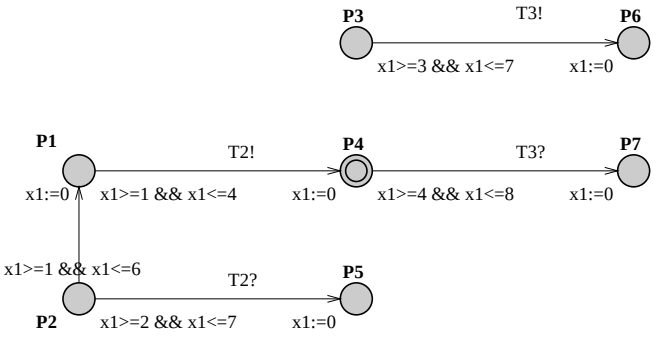


Figure 3.11: Template $P2$ in the Corresponding TAN Model for 2-Conservative TAPN

With reference of the example, we give the formal reduction of 2-conservative TAPN below.

Definition 26. *(Reduction of 2-Conservative TAPN)*
Let $N = (P, T, F, \texttt{times})$ with initial marking M_0 be a 2-conservative TAPN without useless transitions. We define the corresponding timed automata network as a network of timed automata $TAN(N) = A_1(N)|A_2(N)$ where $A_1(N) = (Act_\tau, L^1, l_0^1, C, E^1)$ and $A_1(N) = (Act_\tau, L^2, l_0^2, C, E^2)$ such that

1. $Act_\tau \stackrel{def}{=} \{\tau\} \cup \{\ t! \mid t \in T_2\} \cup \{\ t? \mid t \in T_2\}$,
2. $L^1 = L^2 \stackrel{def}{=} P$,
3. l_0^1 *is defined as the place where the first token in* M_0 *is, while* l_0^2 *is defined as the place where the second in* M_0 *is,*
4. $C \stackrel{def}{=} \{x_1, x_2\}$,
5. *for all the* $t \in T$,
   (a) *if* $t \in T_1$, *we add*
      i. $(l^1, g^1, a, r^1, l^{1\prime}) \in E^1$ *where*
         - $l^1 \stackrel{def}{=} p$ *s.t.* $\{p\} = {}^\bullet t$,
         - $g^1 \stackrel{def}{=} \pi_1(p,t) \le x_1 \wedge x_1 \le \pi_2(p,t)$ *where* $[\pi_1(p,t), \pi_2(p,t)] = \texttt{times}(p,t)$ *where* $\{p\} = {}^\bullet t$,
         - $a \stackrel{def}{=} \tau$ *is the internal action,*
         - $r^1 \stackrel{def}{=} \{x_1\}$, *and*
         - $l^{1\prime} \stackrel{def}{=} p'$ *s.t.* $\{p'\} = t^\bullet$.
      ii. $(l^2, g^2, a, r^2, l^{2\prime}) \in E^2$ *where*
         - $l^2 \stackrel{def}{=} p$ *s.t.* $\{p\} = {}^\bullet t$,
         - $g^2 \stackrel{def}{=} \pi_1(p,t) \le x_2 \wedge x_2 \le \pi_2(p,t)$ *where* $[\pi_1(p,t), \pi_2(p,t)] = \texttt{times}(p,t)$ *where* $\{p\} = {}^\bullet t$,
         - $a \stackrel{def}{=} \tau$ *is the internal action,*
         - $r^2 \stackrel{def}{=} \{x_2\}$, *and*
         - $l^{2\prime} \stackrel{def}{=} p'$ *s.t.* $\{p'\} = t^\bullet$.
   (b) *if* $t \in T_2$, *first we fix* $\{p_1, p_2\} = {}^\bullet t$ *and* $\{p_3, p_4\} = t^\bullet$, *then we add*
      i. $(p_1, g^1, t!, r^1, p_3) \in E^1$ *and* $(p_2, g^2, t?, r^1, p_4) \in E^1$ *s.t.*
         - $g^1 \stackrel{def}{=} \pi_1(p_1,t) \le x_1 \wedge x_1 \le \pi_2(p_1,t) \models \texttt{tt}$ *where* $[\pi_1(p_1,t), \pi_2(p_1,t)] = \texttt{times}(p_1,t)$,
         - $g^2 \stackrel{def}{=} \pi_1(p_2,t) \le x_1 \wedge x_1 \le \pi_2(p_2,t) \models \texttt{tt}$ *where* $[\pi_1(p_2,t), \pi_2(p_2,t)] = \texttt{times}(p_2,t)$,
         - $r^1 \stackrel{def}{=} \{x_1\}$, *and*
      ii. $(p_1, g^1, t!, r^2, p_3) \in E_2$ *and* $(p_2, g^2, t?, r^2, p_4) \in E_2$ *s.t.*

- $g^1 \stackrel{def}{=} \pi_1(p_1,t) \leq x_2 \wedge x_2 \leq \pi_2(p_1,t) \models \texttt{tt}$ *where* $[\pi_1(p_1,t), \pi_2(p_1,t)] = \texttt{times}(p_1,t)$,
- $g^2 \stackrel{def}{=} \pi_1(p_2,t) \leq x_2 \wedge x_2 \leq \pi_2(p_2,t) \models \texttt{tt}$ *where* $[\pi_1(p_2,t), \pi_2(p_2,t)] = \texttt{times}(p_2,t)$,
- $r^2 \stackrel{def}{=} \{x_2\}$.

Theorem 6. *2-conservative TAPNs are polynomial time reducible to timed automata network(s), preserving reachability, i.e., for a marking M of a 2-conservative TAPN N with the initial marking M_0, we have*

$$M \in [M_0\rangle \text{ iff } \forall C \in f(M_0).\ \exists C' \in f(M) \text{ s.t. } C \rightarrow^* C'. \tag{3.5}$$

Proof. The proof idea could be explained in two parts, according to two kinds of different transitions in the net. For those transitions $t \in T_1$, the arguments are the same as that of the 1-conservative TAPN. For those transitions $t \in T_2$, it is more complex. Synchronization between different processes is critical.

First, let us look at the case that M' is reachable in one step from M, i.e., M' is reachable from M by firing a transition t or a time elapsing transition $\epsilon(d)$. We will prove that

$$M[t\rangle M' \text{ iff } \forall C \in f(M).\ \exists C' \in f(M') \text{ s.t. } C \xrightarrow{\tau} C', \tag{3.6}$$

and

$$M[\epsilon(d)\rangle M' \text{ iff } \forall C \in f(M).\ \exists C' \in f(M') \text{ s.t. } C \xrightarrow{\epsilon(d)} C'. \tag{3.7}$$

Let $N = (P, T, F, \texttt{times})$ with initial marking M_0 be a 2-conservative TAPN without useless transitions, and $TAN(N) = A_1(N)|A_2(N)$ is the corresponding timed automata network as defined above.

1. $M[t\rangle M' \Rightarrow \forall C \in f(M).\ \exists C' \in f(M')$ s.t. $C \xrightarrow{\tau} C'$
   Let $f(M) = \{(p_1, p_2, u_1), (p_2, p_1, u_2)\}$ and $f(M') = \{(p_1', p_2', u_1'), (p_2', p_1', u_2')\}$
   (a) $t \in T_1$, let ${}^\bullet t = \{p\}$ and $t^\bullet = \{p'\}$, thus either
      i. $p = p_1$
      For all $C \in f(M)$, we have $(p_1, p_2, u_1) \xrightarrow{\tau} (p', p_2, \bar{u_1})$ and $(p_2, p_1, u_2) \xrightarrow{\tau} (p_2, p', \bar{u_2})$.
      According to the Definition 15, we have
      $\bar{u_1}(x_1) = 0, \bar{u_1}(x_2) = u_1'(x_2)$ and $\bar{u_2}(x_1) = u_2'(x_1), \bar{u_2}(x_2) = 0$,
      hence $(p', p_2, \bar{u_1}) \in f(M')$ and $(p_2, p', \bar{u_2}) \in f(M')$.
      ii. $p = p_2$ is symmetric to case 1(a)i.
   (b) $t \in T_2$, let ${}^\bullet t = \{p_1, p_2\}$ and $t^\bullet = \{p_1', p_2'\}$, thus either

i. $(p_1, p_2, u_1) \xrightarrow{\tau} (p_1', p_2', \bar{u_1})$ or $(p_1, p_2, u_1) \xrightarrow{\tau} (p_2', p_1', \bar{u_2})$
According to Definition 15,
$\bar{u_1}(x_1) = 0 = u_1(x_1), \bar{u_1}(x_2) = 0 = u_1(x_2)$
$\bar{u_2}(x_1) = 0 = u_2(x_1), \bar{u_2}(x_2) = 0 = u_2(x_2)$,
and hence $(p_1', p_2', \bar{u_1}) \in f(M')$, $(p_2', p_1', \bar{u_2}) \in f(M')$.

ii. (p_2, p_1, u_2) is symmetric to case 1(b)i.

2. $M[\epsilon(d)\rangle M' \Rightarrow \forall C \in f(M).\ \exists C' \in f(M')$ s.t. $C \xrightarrow{\epsilon(d)} C'$
Let $f(M) = \{(p_1, p_2, u_1), (p_2, p_1, u_2)\}$, so $f(M') = \{(p_1, p_2, u_1'), (p_2, p_1, u_2')\}$.

(a) $(p_1, p_2, u_1) \xrightarrow{\epsilon(d)} (p_1, p_2, \bar{u_1})$
because
$\bar{u_1}(x_1) = u_1(x_1) + d = u_1'(x_1)$ and
$\bar{u_1}(x_2) = u_1(x_2) + d = u_1'(x_2)$,
therefore $(p_1, p_2, u_1) \xrightarrow{\epsilon(d)} (p_1, p_2, u_1')$

(b) $(p_2, p_1, u_2) \xrightarrow{\epsilon(d)} (p_2, p_1, \bar{u_2})$
because
$\bar{u_2}(x_1) = u_2(x_1) + d = u_2'(x_1)$ and
$\bar{u_2}(x_2) = u_2(x_2) + d = u_2'(x_2)$,
therefore $(p_1, p_2, u_2) \xrightarrow{\epsilon(d)} (p_1, p_2, u_2')$

3. Let M, M' be markings of N, $f(M) = \{(p_1, p_2, u_1),\ (p_2, p_1, u_2)\}$ and $f(M) = \{(p_1', p_2', u_1'),\ (p_2', p_1', u_2')\}$, we show that

$$\forall C \in f(M).\ \exists C' \in f(M') \text{ s.t. } C \xrightarrow{\tau} C' \ \Rightarrow\ M[t\rangle M'.$$

Since we know that either $(p_1, p_2, \bar{u_1}) \xrightarrow{\tau} (p_1', p_2', u_1')$ or $(p_1, p_2, \bar{u_1}) \xrightarrow{\tau} (p_2', p_1', u_2')$ must exist, then

(a) let $(p_1, p_2, u_1) \xrightarrow{\tau} (p_1', p_2', u_1')$, due to Definition 26, this is possible because of some $t \in T$.

i. $t \in T_1$

A. ${}^\bullet t = \{p_1\}$
$p_2' = p_2, \{p_1'\} = t^\bullet, u_1'(x_1) = 0, u_1'(x_2) = u_1(x_2)$,
and hence $M[t\rangle M'$;

B. ${}^\bullet t = \{p_2\}$ is symmetric.

ii. $t \in T_2$, so according to Definition 15 and Definition 26, ${}^\bullet t = \{p_1, p_2\}, t^\bullet = \{p_1', p_2'\}$,
thus $u_1'(x_1) = u_1'(x_2) = 0$, i.e., $M'(p_1) = M'(p_2) = \{0\}$,
and hence $M[t\rangle M'$.

(b) $(p_1, p_2, u_1) \xrightarrow{\tau} (p_2', p_1', u_2')$ is symmetric to case 3a.

4. For the case

$$\forall C \in f(M).\ \exists C' \in f(M') \text{ s.t. } C \xrightarrow{\epsilon(d)} C' \ \Rightarrow\ M[\epsilon(d)\rangle M',$$

the time elapsing steps are obvious as before.

With all of above, Equation (3.6) and Equation (3.7) are proved. Now we show the proof of Equation (3.5) by induction.

Equation (3.5) can be also written in the way

$$M_0[t_1\rangle M_1[t_2\rangle \cdots [t_n\rangle M' \text{ iff } \forall C \in f(M_0).\ \exists C' \in f(M') \text{ s.t. } C \rightarrow^* C'. \quad (3.8)$$

1. $n = 0$, it is obvious that any marking is reachable from itself.
2. $n = 1$,

   $$M_0[t_1\rangle M' \text{ iff } \forall C \in f(M_0).\ \exists C' \in f(M) \text{ s.t. } C \rightarrow^* C'$$

   is obviously true from what we proved before.
3. Assume that Equation (3.8) is true when $n = i$:

   $$M_0[t_1\rangle M_1[t_2\rangle \cdots [t_i\rangle M' \text{ iff } \forall C \in f(M_0).\ \exists C' \in f(M') \text{ s.t. } C \rightarrow^* C'. \quad (3.9)$$

   then for the case $n = i + 1$,
   we have $M_0[t_0\rangle M_1[t_1\rangle \cdots [t_i\rangle M'[t_{i+1}\rangle M''$ on the left side.
   From Equation (3.6) and Equation (3.7), we know that

   $$M'[t_{i+1}\rangle M'' \text{ iff } \forall C' \in f(M').\ \exists C'' \in f(M'') \text{ s.t. } C' \rightarrow^* C''. \quad (3.10)$$

   So from Equation (3.9) and Equation (3.10) we have

   $$M_0[t_0\rangle M_1[t_1\rangle \cdots [t_i\rangle M'[t_{i+1}\rangle M'' \text{ iff } \forall C \in f(M_0).\ \exists C'' \in f(M'') \text{ s.t. } C \rightarrow^* C''.$$

With all of above, Equation (3.5) is proved.

□

Remark 5. *For a 2-degree TAPN, we can translate it into network of TA according to the same technique which is used in the reduction of 2-conservative TAPN, and the number of tokens will correspond to the number of parallel components.*

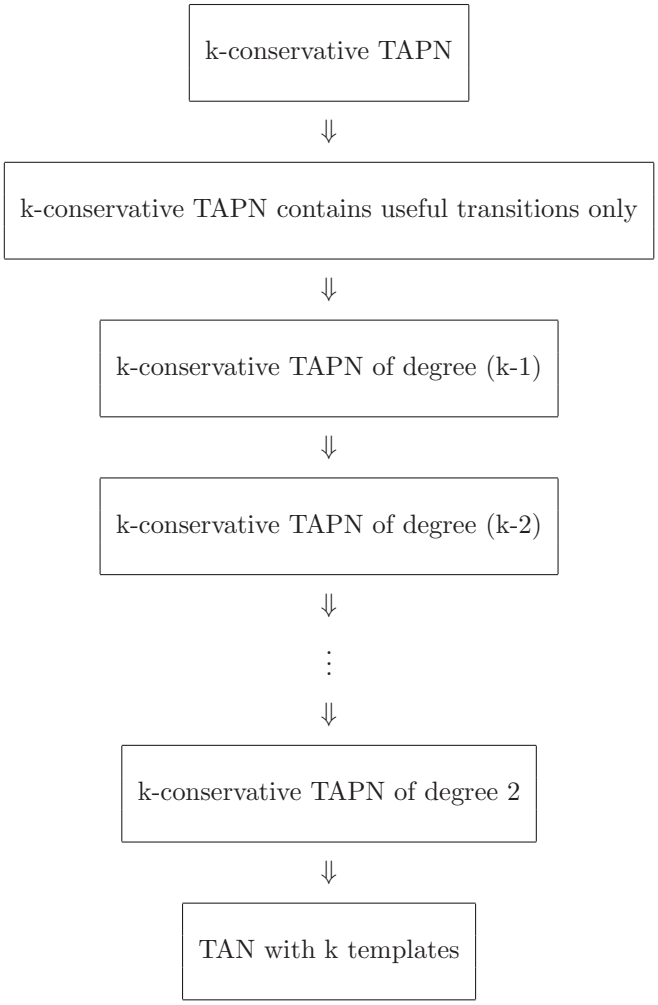
So in this case, we can translate all the k-conservative TAPN into TA/TAN by synthetically using the reduction of degree and from TAPN to TA/TAN.

3.5 Conclusion

In this chapter, we discussed the reduction techniques from k-conservative TAPN to TA/TAN. Generally, we translate a given TAPN by

1. Removing useless transitions.
2. Reducing the degree from k to k-1, until we get a 2-degree TAPN.
3. Reducing the TAPN to TA/TAN.

For 1-conservative TAPN, we can directly translate it into TA after we moved the useless transitions. While for those k-conservative TAPN, we follow the way shown below:



4 Implementation

4.1 Introduction

In this chapter we emphasize on the implementation of our reduction algorithm. The idea behind the implementation is that we would like to modify a given XML file of TAPN format into the XML format which is valid for the UPPAAL tool. The input XML file satisfies the DTD [39] format what we have proposed for a TAPN. Our aim is to read the input XML which is a TPAN and process it and generate UPPAAL compatible XML file as output. By doing it so we can actually verify the reachability of the newly generated TA which corresponds to the original TAPN. As our main emphasis is to find a way to automate the process so that it can be used for reachability of TAPN.

We have started using the XML and Java technologies as the implementation more specifically we need XML (extensible markup language) the format used to store the TAPN and TA diagrams. In order to read and write these XML file we have chosen Java programming language because of its portability and also we have various ready made API such as SAX [33], DOM [14] and JDOM [22].

4.2 Tools and Programming Languages Used

In this section we give brief description about the various tools and programming languages we have used in developing the parser.

4.2.1 XML

Extensible Markup Language (XML) is simple and convenient text format derived from SGML (ISO 8879).XML is originally designed for the publication of large scale electronic data. XML is playing a crucial role in the exchange of a wide variety of data on the web. It is one of the extensions to the HTML where the tags are predefined. Most importantly XML is W3C recommended form of data storage and representation.

4.2.2 Why XML?

Since there are many data storage techniques and software tools available one would wonder why we have chosen XML as our standard data storage and manipulating tool. The answer is very simple and straight forward, we want the data to be re-used and manipulated rather than confining it to remain dormant. Unlike HTML where the data presentation is given more importance, XML gives means to store, carry and exchange data.

To add further XML has the versatility to create user defined tags while creating the data representation format. The basic condition before creating any XML file that it has to follow DTD format. For this reason we have agreed on the DTD of TAPN and followed that format right through the parser implementation. Unlike the other data storage tools like MS-Access or Oracle where data is stored in the form of tables in XML data is stored in the form of XML files. And we have so many data accessing tools for these XML files. Also, the data we need to store isn't very large which has to be stored putting some constraints while accessing and writing data. We were constantly looking for a tool which could provide us simplicity in storing and retrieving data, apart from the that more flexible in exchange of data. XML was the good choice for us when compared with other existing tools.

4.2.3 Java

We have opted to use the Java programming language to parse the XML documents. Its because of its portability. We use JAVA to implement the concrete programming. Because in this project, what are more focussed in finding a way for the problem and translation here is not the bottle neck. In this case, a high-level and highly modularized programming tool is a good choice.

In this process we have considered various Java parsers like JDOM, SAX etc and decided to go with the JDOM (Java Document Object Model). The XML JDOM is Java enabled API. We will explain few more things regarding this JDOM in next few sections. Based on the XML objects which need to be modified, we use XMLSpy [42], which is a powerful XML file editing tool of Altova Inc.. Altova XMLSpy 2004 Home Edition is an entry level XML development tool for designing and editing applications involving XML technologies.

4.2.4 JDOM

The JDOM is an Application Programming Interface (API) for HTML and XML documents. As a W3C specification, the objective for the XML JDOM has been to provide a standard programming interface to a wide variety of applications. The XML JDOM is designed to be used with most programming languages and operating systems. With the XML JDOM, a programmer can create an XML document, navigate its structure, and add, modify, or delete its elements. Java Document Object Model(JDOM) is a specification for a set of interfaces that XML parser which can be implemented in order to provide a model of an XML document as set of objects.

There are two main parts to the JDOM:

1. the core that describes interfaces for accessing XML, and
2. the HTML specific JDOM API.

The JDOM represents a tree view of the XML document. The `documentElement` is the top-level of the tree. This element has one or many `childNodes` that represent the branches of the tree. A *Node Interface Model* is used to access the individual elements in the node tree. As an example, the `childNodes` property of the `documentElement` can be accessed with a for each construct to enumerate each individual node.

XML elements can be extracted from an XML document by traversing the node tree, by accessing the elements by number, or by accessing the elements by name.

One common way to extract XML elements from an XML document is to traverse the node tree and extract the text value of each element.

4.3 XML Reader and Writer

In this section we would like to describe in few words about the XML Reader and Writer which are the two main parts in the parser. The reader and the writer are written using the JDOM API. We have presented the XML representation format for the TAPN and the TA generated by the parser.

4.3.1 XML Reader

This part particularly represents the document format i.e. the XML format of TAPN and the various methods of the JDOM API used to access the data. Before writing further first we would like to show the XML format of a TAPN showed in Figure 4.1 below.

```
1  <?xml version="1.0"?>
2
3  <tpn>
4
5  <place> <name>P0</name> <pid>pid0</pid> </place>
6  <place><name>P1</name> <pid>pid1</pid> </place>
7
8  <token> <id>pid0</id> </token>
9
10 <transition><id>T1</id> <label>one</label> </transition>
11 <transition> <id>T2</id> <label>two</label> </transition>
12
13 <arc>
14 <source>pid0</source>
15 <target>T1</target>
16 <type>PlaceTransition</type>
17 <lb>5</lb>
18 <ub>500</ub>
19 </arc>
```

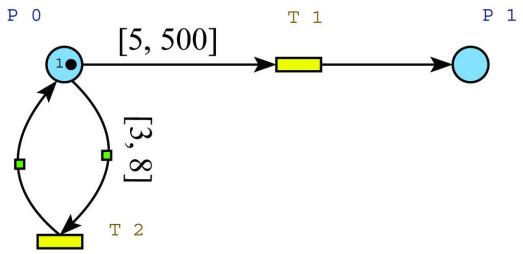


Figure 4.1: A Given Example of TAPN

```
20
21 <arc>
22   <source>T1</source>
23   <target>pid1</target>
24   <type>TransitionPlace</type>
25   <lb>0</lb>
26   <ub>0</ub>
27 </arc>
28
29 <arc>
30  <source>pid0</source>
31  <target>T2</target>
32  <type>PlaceTransition</type>
33  <lb>3</lb>
34  <ub>8</ub>
35  </arc>
36
37   <arc>
38   <source>T2</source>
39   <target>pid0</target>
40   <type>TransitionPlace</type>
41   <lb>0</lb>
42   <ub>0</ub>
43   </arc>
44
45 </tpn>
```

The reader basically reads the data stored in these XML files using the notions. In the following piece of code we show how to read the token id of the TAPN which is important in deciding the start place of the TA. We make use of the List data structure and create the list with elements which are defined with in the `token`. After that we use a iterator to process through those elements and find the data which is encapsulated with in `id` section of the token element and store it to some integer variable. Like this we use different number of lists to various elements and sub-elements and to process through them to retrieve the data.

```
     List tokenList = root.getChildren("token");
     Iterator tokenIterator = tokenList.iterator();

      while (tokenIterator.hasNext()) {

       Element tokenElement = (Element) tokenIterator.next();
       id=String.valueOf(tokenElement.getChild("id").getTextTrim())
           ;
     }//end of while loop
```

4.3.2 XML Writer

In XML writer the predefined functions of JDOM like the `setText()`, and `setIndex()` comes in handy while constructing the XML document. The following code represents the insertion of the *declaration* element with some data into it. The sub elements are added into the main element in the similar manner.

```
    public void setDeclaration(String app3Name) {
        Element setDeclaration = new Element("declaration");
        setDeclaration.setText(app3Name);
        insertBefore("template",setDeclaration);
    }
```

Unlike the other built in functions the `insertBefore()` function searches the possible children for the given elements and inserts the current element at appropriate place. Before inserting it ensures the place of the element to be inserted. The `insertBefore()` function is given below.

```
private Element insertBefore(String tag, Element w_element){

    List w_children = doc2.getRootElement().getChildren();
    if(w_children.isEmpty()){
        doc2.getRootElement().addContent(w_element);
    }
    else {
        int position = 0;
        Iterator it = w_children.iterator();
        while (it.hasNext()) {
            if(((Element)it.next()).getName().equals(tag)) {
                break;

            }//end of if condition

            position++;

        } //end of while loop

        w_children.add(position, w_element);

    }//end of else

    return w_element;

    } // end of insertBefore
```

4.3.3 XML Output

The following XML represents the XML file of TA generated by the parser which is UPPAAL compatible, illustrated as in Figure 4.2.

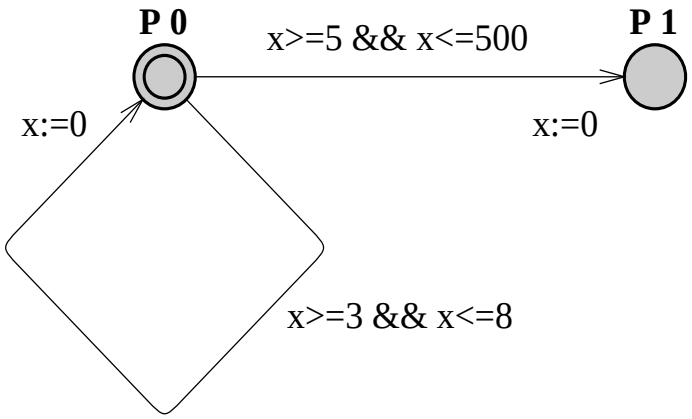


Figure 4.2: Output TA of the Given Example of TAPN

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE nta PUBLIC
"-//Uppaal Team//DTD Flat System 1.0//EN"
"http://www.docs.uu.se/docs/rtmv/uppaal/xml/flat-1_0.dtd">

<nta>
  <declaration>chan T1,T2</declaration>
  <template>
    <name>P1</name>
    <parameter />
    <declaration>clock x;</declaration>
    <location id="pid0">
      <name>P0</name>
    </location>
    <location id="pid1">
      <name>P1</name>
    </location>
    <init ref="pid0" />
    <transition>
      <source ref="pid0" />
      <target ref="pid1" />
      <label kind="guard">x&gt;=5 and x&lt;=500</label>
      <label kind="assignment">x:=0</label>
      <label kind="synchronisation" />
    </transition>
    <transition>
      <source ref="pid0" />
      <target ref="pid0" />
      <label kind="guard">x&gt;=3 and x&lt;=8</label>
      <label kind="assignment">x:=0</label>
      <label kind="synchronisation" />
    </transition>
  </template>
  <template>
    <name>P2</name>
```

```
35      <parameter />
36      <declaration>clock x;</declaration>
37    </template>
38    <instantiation>//intialise here</instantiation>
39    <system>system P1,P2;</system>
40  </nta>
```

4.4 Reduction Algorithms

We have the reduction algorithms followed in implementing the parser. The whole parser is implemented following Theorem 5, Theorem 6 and Theorem 2 shown before.

For Theorem 2, we implemented a parser that could able to convert the 3-conservative TAPN to 2-conservative TAPN. The idea behind this is to convert the input XML document which is in 3-conservative TAPN form into 2-conservative TAPN XML format. Then follow Theorem 6 and Theorem 5, we have implemented a parser that can convert the conservative TAPN of degree 1 and 2 into TA XML format which is given to the UPPAAL as input and is valid as well.

Packages of full programs codes are available in the attached CD.

5 Case Study

In this chapter we will show how real-world examples can be modeled as timed-arc Petri nets. We will model Fischer's protocol [41] which is one of the solutions to mutual exclusion problem and Alternating Bit Protocol [5] which shows the way how we communicate bit data in network.

5.1 Case Study I: Fischer's Protocol

Now, let us have a look at a good practical application of translating a 3-conservative TAPN into timed automata.

5.1.1 Motivation and Application

For security reasons, we sometimes want to avoid having more than one process available for a certain event at a certain time, like the two sides of a coin.

Once we have available to use a mechanism to provide synchronization, where update is thereby sequentially ordered, we say that the processes provide *Mutual Exclusion* over a shared resource. *Mutual Exclusion* is the assurance that only one process is given access to a shared resource at any one time [10].
It is a good case for us to build a TAPN model, and more important to verify the properties of the reduced TAN through tool Uppaal.

5.1.2 Fischer's Protocol

Why Fischer's Protocol

We choose Fischer's Protocol as a case study for this *Mutual Exclusion* problem. The protocol was first proposed by Fischer [16], and later studied in [2, 23, 25, 34]. Instead of using atomic test-and-set instructions or semaphores, as is nowadays often done to assure mutual exclusion, Fischer's protocol only assumes atomic reads and writes to a shared variable. Mutual exclusion in Fischer's Protocol is guaranteed by carefully placing bounds on the execution times

of the instructions, leading to a protocol which is very simple, and relies heavily on time aspects. This makes it an ideal candidate for the purpose we have in mind, namely to try to verify a not too difficult protocol which still has quite intricate timing aspects.

How Does It Work

The idea behind the protocol is that the timing constraints on the local clocks are set so that only one process can enter the critical section if the shared variable is equal to its own number.

Assume a concurrent system with n processes $P_1, \cdots P_n$. Let x_i be the local clock for each process P_i. Then the formal description of it is given as following:

- $P_i \stackrel{def}{=} A_i$
- $A_i \stackrel{def}{=} (\{v = 0\}, \tau, \{x_i\}).B_i$
- $B_i \stackrel{def}{=} (\{x_i < \mathbf{const}_1\}, \tau, \{v := i, x_i\}).C_i$
- $C_i \stackrel{def}{=} (\{v = i, x_i > \mathbf{const}_2\}, \tau, \{v := i\}).CS_i$
  or $(\{v \neq i, x_i > \mathbf{const}_2\}, \tau, \{\}).A_i$
- $CS_i \stackrel{def}{=} (\{v = i\}, \tau, \{v := 0\}).A_i$.

Figure 5.1 illustrates the protocol intuitively. Each process P_i may be in either of the four local states A_i, B_i, C_i, CS_i. At first, all processes are in the A-state and the shared variable v is initially 0. Then a process P_i which tries to enter the critical section will change state from A_i to B_i if it sees $v = 0$. After then, it will move on from B_i to C_i before the clock x_i proceeds to $\mathbf{const}_1$, meanwhile reset the clock x_i to 0 and assign v to its own process number i. From C_i, when the clock value of x_i is larger than $\mathbf{const}_2$, it can move to the critical section CS_i if v is still equal to its own process number, or go back to the A-state if v is not equal to its own process number. Once a process P_i enters the critical section, it can move back to A-state and assign v to 0, to start a new round.

5.1.3 Modeling

In this section, we take the example of avoiding two processes reach the critical section at the same time. Which means to prove the actual mutual exclusion between the two critical sections.

We model the system of two processes as a 3-conservative TAPN (see Figure 5.2) and then automatically convert it into a TAN, and then verify its properties using the verification tool UPPAAL.

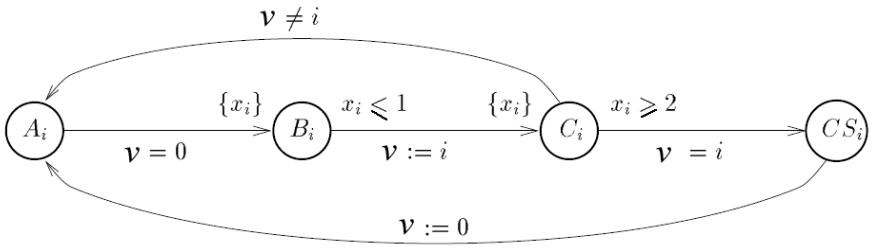


Figure 5.1: Fischer's Protocol

Each process path has a resource request place (i.e. A_i), a waiting place (i.e. B_i), a controlling place (i.e. C_i) and a critical section place (i.e. CS_i). We have considered each token as process (we use token instead of process hereafter) and one of the places as critical section (we use critical place instead of critical section hereafter) in the net so that at a given time only one of the tokens remain in that place. We model the shared variable with three controlling places (i.e. v_i where $i = 0, 1, 2$), which always contain the third token in one of them and help the token in each process to reach the critical place. There is one condition to be satisfied by the process before entering the critical place. For any token before entering the critical place should have the message token within its controlling state. If the above condition satisfies then only it can actually enter the critical place.

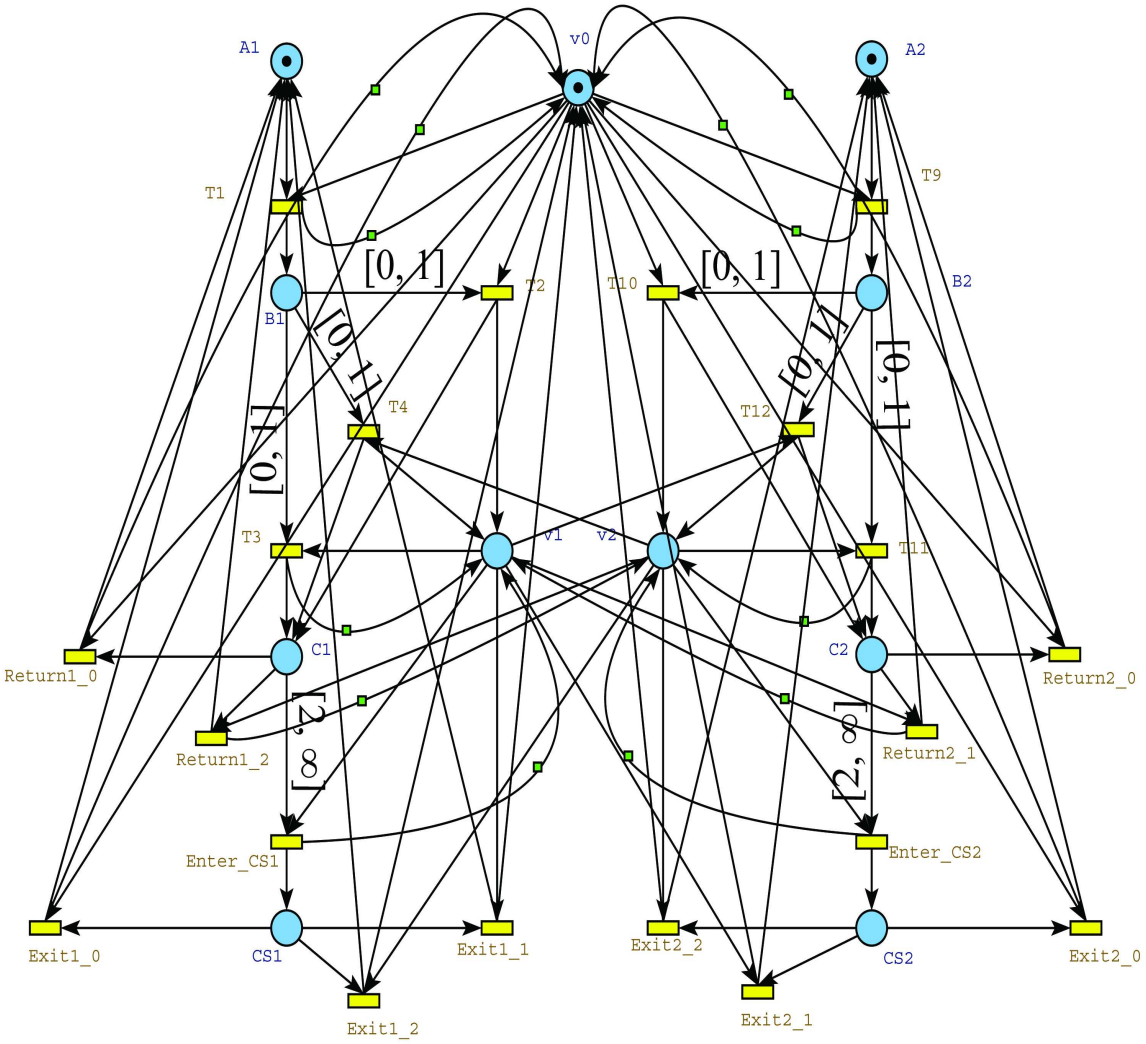


Figure 5.2: TAPN for Fischer's Protocol with 2 Processes

5.1.4 Verification

To know the correctness of our reduction, we verify the automatically generated TAN in UPPAAL. Because we model the TAPN according to the protocol itself, so if it outputs the expected results for this corresponding TAN, then we can conclude that the protocol is correct.

Actually, we have verified the Fischer's protocol using the reduction we present before. In the 3-conservative TAPN, a third token is used to model the shared variable and as a whole abiding the Fischer's protocol design. We have considered two processes trying to enter the critical places named CS_1 and CS_2, respectively. Only one of the two can reach the goal.

The protocol is not safe if there are tokens in both CS_1 and CS_2 at some certain time, while no matter what the age of each token is. We can easily verify the possibility of this case in the reduced TAN, i.e., to check if both of the corresponding locations CS_1 and CS_2 can be reached at some time.

If we have a dry run of the net we can see the first token which actually enters the resource request place will remain the last to enter the critical place which is characteristic of this protocol. Our aim is to verify this automatically.

With the tool UPPAAL, we verified the reachability in the reduced TAN. To verify the reachability with a loose constraint on time, we omit clock constraint in the verifier formula in UPPAAL, i.e., we only have to verify the reachability or unreachability of any certain location(s). Figure 5.3 illustrates one of the templates of the reduced time automata.

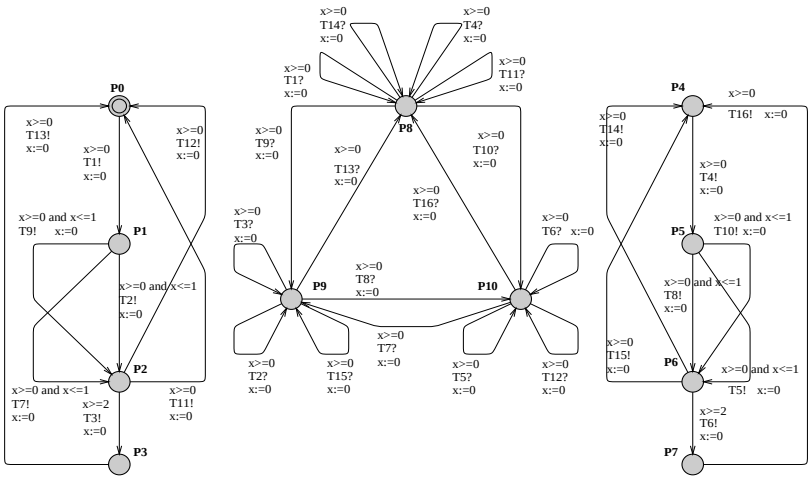


Figure 5.3: Template 1 in the TAN for Fischer's Protocol with 2 processes, Template 2 and Template 3 are the same except with the initial location $P4$ and $P8$, respectively.

And then we have the verification results as below:

1. `E<>Token1.CS1 or Token2.CS2`
   Property is satisfied.

2. `E<>Token1.CS1 && Token2.CS2`
   Property is not satisfied.

Query 1 verifies the reachability of the model, i.e., either *process 1* or *process 2* can reach the critical section. And Query 2 verifies the unreachability of the model, i.e., the situation which both *process 1* and *process 2* reach the critical sections cannot happen. In other words, the model implements Fischer's Protocol for Mutual Exclusive problem.

Further more, we construct a *Error-Check* TAPN for the original model, which contains three extra *error-check* transitions (with degree 3) and three extra places which represent the error unsafe property of the model. Figure 5.4 illustrates this modified TAPN with degree 3. For simplification reasons, we present only the part which the newly added places and transitions involved and the ignored part keeps the same as the original TAPN. With the reduction technique, we also get the reduced TAN (one of the templates Token1 is illustrated in Figure 5.5) and check the reachability of the TAN with tool UPPAAL. The verifier shows correct results as below, i.e., the correctness of our reduction techniques.

1. `E<>Token1.Error or Token2.Error or Token3.Error`
   Property is not satisfied.

In this modified model, Query 1 verifies the unreachability or the *Error* location, i.e., there cannot be more than one process can reach the critical section.

There is not the possibility that two processes reach their critical sections at the same time, i.e., in the original TAPN, there cannot be the case that tokens exist in both places CS_1 and CS_2 at the same time. Intuitively, the situation which there is a token in CS_1, a token in CS_2 and the third token in either v_0, v_1 or v_2 is not available.

5.2 Case Study II: Alternating Bit Protocol

In this section, we choose alternating bit protocol [8] as the second discussion of our case studies.

5.2.1 Motivation and Application

Network and communication act more and more important roles in the IT industry and human life nowadays. To make sure that the information is transferred correctly is one of the most important goals over the network. Many communication protocols make important use of timing values in their specifications,

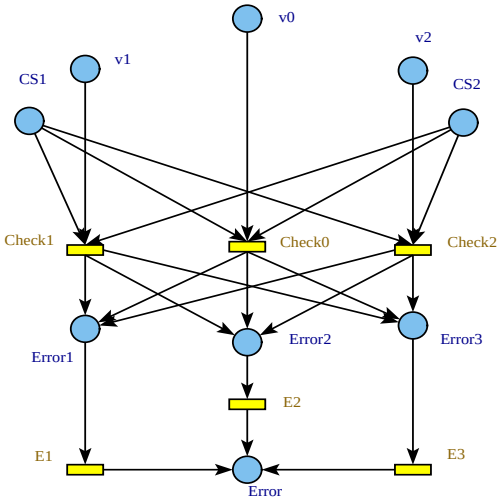


Figure 5.4: Modified TAPN for Fischer's Protocol, with Error-check Places

such as recovery from losses of messages is implemented using time-outs. And alternating bit protocol is one of them which provides an introductory example to the analysis method.

5.2.2 Alternating Bit Protocol

Why Alternating Bit Protocol

Alternating bit protocol was brought first by K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson in 1969. The protocol transmits messages between two participators, allowing only one message being transmitted at a time, over an unreliable transmission medium [8]. On the behavior of the transmission medium, the protocol assumes that messages or acknowledgments may be lost in transit. Particularly, Bernard Berthomieu and Michel Diaz presented a time Petri net (TPN) model for this protocol in [5]. Thus it will be a good object for our second case study.

How Does It Work

The protocol is based on such a idea that the recovery from losses is done by using a time-out and retransmitting mechanism. i.e., the sender records the time when it sends a message and if an acknowledgment of its delivery does not return within a given time, then the message is retransmitted.

The text below is partly cited from [5]. This selected mechanism must be suf-

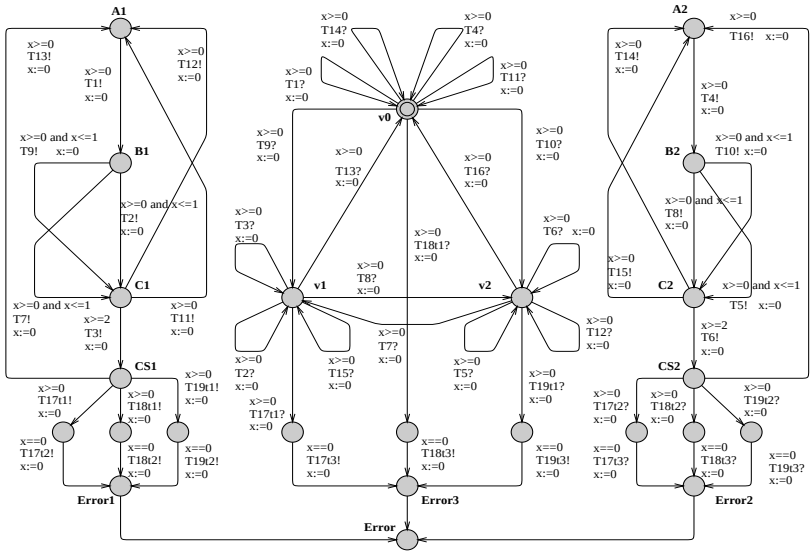


Figure 5.5: Template Token1 of the Modified TAN for Fischer's Protocol, with Error-check Places

ficient for recovering from losses and for preventing the acceptance of duplicate messages. i.e., upon the reception of a message, the receiver must be able to decide whether this message is a new message or an old one (duplicate). To solve this problem, messages are numbered, prior to transmission, with modulo-2 sequence numbers and, for every packet received, an acknowledgment is sent that carries the sequence number of the received packet.

5.2.3 Modeling

Our model is inspired by a similar model in [5]. The losses of messages and acknowledgments are represented simply as transitions without output places. In other words, a token will be consumed once such a transition is fired, i.e. the message or acknowledgment is lost then. In the meanwhile, time-out and re-transmitting mechanisms are represented by the transitions which have different time constraints.

We modify the model as a conservative TAPN with degree 2 (see Figure 5.6). Basic idea is that adding an extra place *Buffer* which contains the extra token/-tokens and keep the conservative property of the net.

The meanings of the transitions are as follows:

SP0	Send Packet 0
RP0	Resend Packet 0
RA0	Receive Acknowledgment 0
RLP0	Receive and Release Packet 0
SA0	Send Acknowledgment 0
RJP0	Receive and Reject Packet 0
LP0	Lose Packet 0
LA0	Lose Acknowledgement 0
SP1	Send Packet 1
RP1	Resend Packet 1
RA1	Receive Acknowledgment 1
RLP1	Receive and Release Packet 1
SA1	Send Acknowledgment 1
RJP1	Receive and Reject Packet 1
LP1	Lose Packet 1
LA1	Lose Acknowledgement 1

We first start with no timing information, thus we have the TAPN model (see Figure 5.6) with loose time constraints (i.e. all transitions have time constraints $[0, \infty]$) and then we are able to check the reachability of the reduced TAN in UPPAAL.

5.2.4 Verification and Modified Model

According to the protocol, it is not safe if

1. the *Receiver* sends the acknowledgement while the *Sender* has not sent it yet, nor

2. the *Sender* is sending the next bit of message while the *Receiver* has not sent the acknowledgement yet.

When the original TAPN is 3-conservative i.e. there is only one token in *Buffer*, we can easily verify the possibilities of above cases in the reduced TAN with the tool UPPAAL. As how we verified in Section 5.1, we omit clock constraint in the verifier formula in UPPAAL, i.e., we only have to verify the reachability or unreachability of any certain location(s). In the verifier of UPPAAL, we have the results as below:

1. `E<>(Token1.S3 && Token2.R1) or (Token1.S1 && Token2.R3)`
   `Property is not satisfied.`

2. `E<>(Token1.S1 or Token1.S3 or Token1.S4) && Token2.R2`
   `Property is not satisfied.`

3. `E<>(Token1.S1 or Token1.S2 or Token1.S3) && Token2.R4`
   `Property is not satisfied.`

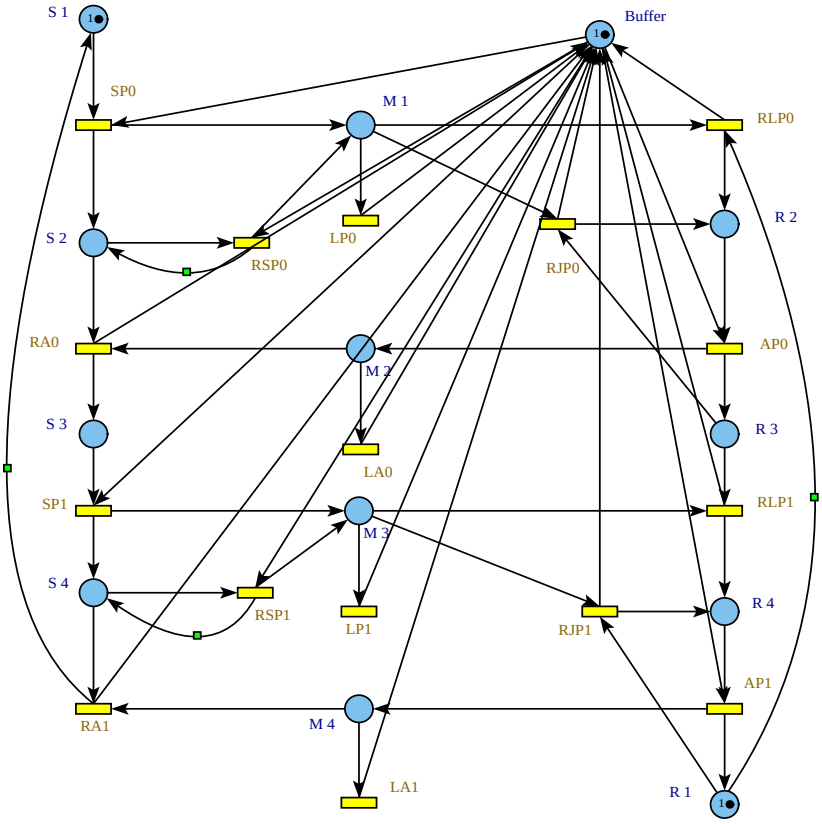


Figure 5.6: TAPN with Loose Time Constraints for Alternating Bit Protocol

Query 1, 2 and 3 verify the unreachability of the model. More precisely, Query 1 verifies the situation of Unsafety 1 (the *Receiver* sends the acknowledgement while the *Sender* has not sent it yet) cannot happen. Query 2 and Query 3 verify the situation of Unsafety 2 (the *Sender* is sending the next bit of message while the *Receiver* has not sent the acknowledgement yet) cannot happen. In other words, the model implements alternating bit protocol.

It seems that we get a correct result as what we should have from the protocol. But the situation is when we add more tokens in *Buffer*, safety of the model is not kept any more.

1. `E<>(Token1.S3 && Token2.R1) or (Token1.S1 && Token2.R3)`
   `Property is satisfied.`

2. `E<>(Token1.S1 or Token1.S3 or Token1.S4) && Token2.R2`
   `Property is satisfied.`

3. `E<>(Token1.S1 or Token1.S2 or Token1.S3) && Token2.R4`
   Property is satisfied.

With further study on it, we find that it is because both of the communication between *Sender* and *Buffer*, and *Receiver* and *Buffer* are easy to be synchronized. Whenever one of the tokens in *Buffer* is consumed by the synchronization with *Sender*, there will still be another token with the right time constraint available for the synchronization with *Receiver*. The problem is the token now in the *Receiver* will be always ready to fire because of the loose time constraints.

In this case, we add some time constraints according to the general case of the protocol. For each output arc of *Buffer* and each input arc of the `Lose` transitions (i.e. `LP0`, `LA0`, `LP1` and `LA1`), we add $[0, \infty]$, i.e., the intervals $[0, \infty]$ are given for sending the numbered messages. For the arcs from *S2* to `RSP0` and from *S4* to `RSP1`, we add $[5, \infty]$. For the other input arcs of a certain transition, we keep the time constraints as what has been given to the transition itself in the TPN model presented in [5]. Thus, we have a modified TAPN model according to the alternating bit protocol itself. As illustrated in Figure 5.7, we can see that it actually is a 3-conservative TAPN with degree 2 only. Similar to that in Section 5.1, the time constraints are regarded as $[0, \infty]$ for those input arcs of a transition where there is no intervals shown in the TAPN.

Based on the modified TAPN with general time constraints, we verify the automatically generated TAN in UPPAAL and get the expected results, i.e., the protocol keeps safety. Figure 5.8 illustrates one of the templates of the reduced time automata.

1. `E<>(Token1.S3 && Token2.R1) or (Token1.S1 && Token2.R3)`
   Property is not satisfied.

2. `E<>(Token1.S1 or Token1.S3 or Token1.S4) && Token2.R2`
   Property is not satisfied.

3. `E<>(Token1.S1 or Token1.S2 or Token1.S3) && Token2.R4`
   Property is not satisfied.

5.2.5 Experiments

With further experiments on the cases where there are more tokens in the place *Buffer*, we get more copies of the same template models the buffer tokens in the translating. We have different timing results on the cases, while the protocol remains safe even with larger capacity of communication channels. The reduced TAN has the same number of templates corresponding to the number of tokens in TAPN model.

System Configuration:

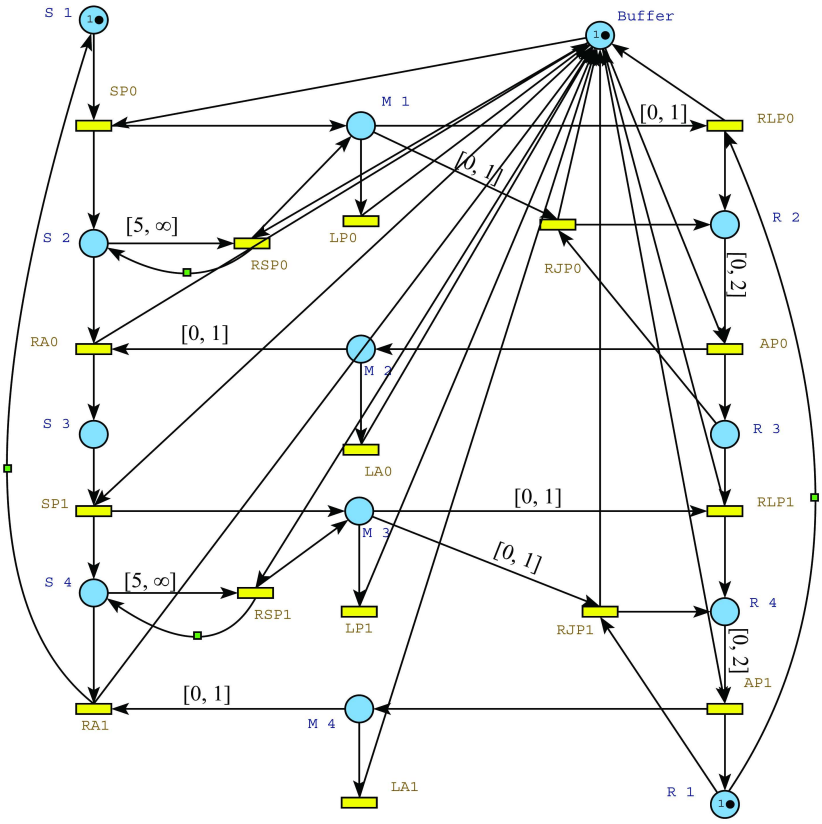


Figure 5.7: Modified TAPN for Alternating Bit Protocol

	Server 1	Server 2
Name	Athlon 8	T4
CPU	AMD Athlon (tm) 700MHz	Pentium M 1.5GHz
RAM	392M	512M
OS	Microsoft Windows 2000 Professional	Microsoft Windows XP Professional

Comparison of the time cost on different number of tokens:

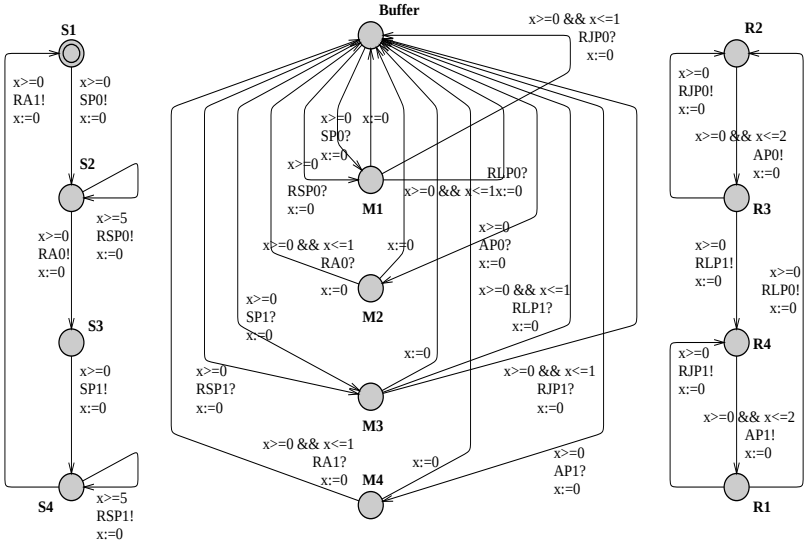


Figure 5.8: Template Token1 of the TAN for Alternating Bit Protocol

Total number of tokens	Time of checking verification property (seconds)	
	Server 1	Server 2
5	<1	<1
6	2-3	1
7	14	11
8	76	74
9	690	616
10	>3600	>3600

According to the results from the table above, we can see that the time cost increases rapidly. We presume that it is because of the rapid increase of the synchronization channels and the explosion of the state space when a template is added .

6 Conclusion and Future Work

Model Checking is becoming more and more inevitable in designing real-time systems because it is important to check the safety property of such systems. It is no longer confined in verifying the hardware as the production of a correct software is equally important.

As a modeling tool, Petri nets are popular as they have strong mathematical and conceptual formalism developed over years. TAPN is one of the extensions to these Petri nets with time features. Modeling in TAPN is a more meaningful way to construct the real-time systems as the time features of TAPN can be extended to model the time features in the real-time systems. The time features in TAPN concentrate on tokens rather than on transitions as in TPN. Thus for some systems, TAPN provides a better way to model.

However, verification tool which is specified for TAPN model is not available to the best of our knowledge. The unavailability of such a specific tool inspired us to find a solution for the problem. As we can see there are two ways to solve this problem, either to develop an independent TAPN verification tool or make uses of existing verification tools by converting the TAPN as the input for such tools.

Since Uppaal, a matured verification tool for timed automata (TA), has been developed for more than ten years and has an optimized high-level efficiency, we choose the second way. We focus on using a reduction technique to translate the input TAPN model into a TA and verify the property in Uppaal.

6.1 Our Contribution

During the thesis work, we studied several subclasses of TAPN and presented formal reduction techniques for converting them from timed-arc Petri nets into timed automata or timed automata network. Relationship between TAPN and TA is built up during the course. The general idea is based on two points:

1. For all conservative TAPN of k degree where k is greater than 2, we reduce the degree to k-1, until we get an equivalent net of degree 2;

2. For 1-conservative TAPN and 2-conservative TAPN, we reduce them directly into TA/TAN.

All the presented reductions preserve reachability. We have further implemented these formal techniques in Java. First a DTD is designed for general TAPN which we have used in the parser. And then our reduction algorithms are implemented as a Java-XML parser which automatically translates the XML file in TAPN format to the UPPAAL compatible XML file. The parser includes both the degree-reduction and the transformation to TA.

We outlined two case studies, Fischer's protocol and alternating bit protocol. Both of them are modeled as TAPNs first and then reduced to the TA/TAN using the implemented parser. The newly generated network of TA are verified in the UPPAAL to check the reachability and safety properties of the protocols. We have got correct answers while verifying the properties of these protocols as expected.

Furthermore, from the results we got in the experiments, we notice that UPPAAL performs better than Roméo in verifying the properties of the model with similar structures.

6.2 Future Work

The idea of reduction from TAPN to TA has been presented in our thesis for some of the subclasses of TAPN, which are the k-conservative TAPN more precisely. However, we only brought one of the possibility of which we can verify some certain properties of a k-conservative TAPN, and built up the relationship between TAPN and TA. It would be interesting to extend the reduction techniques formally to the whole subclass of bounded TAPN. And it would be also interesting that to investigate whether our reductions can be adopted for stronger equivalence notions (e.g. to preserve bisimulation).

In the parser so far we have been constructing the input XML file by hand, and next step would be to have a proper and user-friendly GUI so that the input XML files can be generated automatically by the tool. With that we can say that we have a proper tool for verifying the TAPN which would be the first of its kind.

A Tools and Howtos

A.1 Roméo

We use Roméo as an interface to construct the basic models of Petri nets. Below are the steps we took to make this tool run on Windows:

1. Download the latest version of Roméo from here: `http://www.irccyn.ec-nantes.fr/irccyn/d/en/equipes/TempsReel/logs/software-2-romeo&ver=version-2-2.4.2`.
2. Extracted it.
3. Downloaded TCL from here: `http://www.activestate.com/Products/Download/Register.plex?id=ActiveTcl`.
4. Installed TCL.
5. Run `romeo.tcl` from the folder that was created in step 2.

A.2 Uppaal

The current official release is Uppaal 3.4.10 (May 11, 2005). And more information and further download instructions and user guide are also available on the homepage of Uppaal: `http://www.uppaal.com`. Below we cite the instructions from the Uppaal homepage.

A.2.1 Installation Instructions

To download and install (or upgrade to) the current version of Uppaal:

1. Choose the version from the download area: `http://www.it.uu.se/research/group/darts/uppaal/download.shtml#downloads`.

2. Fill in the license agreement and press the `Accept and Download` button.
3. Download the zip-file containing the installation files.
4. Unzip the downloaded zip-file. This should created a number of files, including: `uppaal2k.jar`, `uppaal`, and the directories `bin-Linux`, `bin-SunOS`, `bin-Win32`, and `demo`. The `bin`-directories should all contain the two files `server(.exe)` and `verifyta(.exe)` plus some additional files, depending on the platform. The directory demo should contain some demo files with suffixes `.xml`, and `.q`.
5. Make sure you have the Java version 1.4 (e.g. J2SE Java Runtime Environment) or newer installed and properly configured on your system. The UPPAAL GUI will not run without Java installed. Java for SunOS, Windows95/98/NT, and Linux can be downloaded from `java.sun.com`.
6. To run UPPAAL on Linux or SunOS systems run the startup script named `uppaal2k` (or `uppaal` in the beta 2). To run on Windows95/98/NT systems, just double-click the file UPPAAL2K.JAR.
7. (Optional) Users can join the `Uppaal` mailing list. The mailing list is intended for users of the tool. To join the list, email:
   `uppaal-subscribe@yahoogroups.com`.

   To post to the mailing list, email:
   `uppaal@yahoogroups.com`.

   More information is available on
   `http://groups.yahoo.com/group/uppaal/`.

A.2.2 Java

Users can download Java 2 Platform Standard Edition 5.0 (J2SE 5.0) on
`http://java.sun.com/j2se/1.5.0/download.jsp`.

More user guide and other downloads are also available.

A.3 XMLSpy

Altova XMLSpy® 2005 is the industry standard XML development environment for modeling, editing, debugging, and transforming all XML technologies, then automatically generating runtime code in multiple programming languages. XMLSpy® 2005 is the ultimate productivity enhancer for J2EE, .NET, Eclipse, and database developers that need the latest XML, Web services and database technologies. More information and free trials could be found on:
`http://www.altova.com/download_spy_enterprise.html`.

B DTD

B.1 Analysis and Diagram

Figure B.1 illustrates the structure of the DTD we use in an input TAPN file, using the GUI(Graphical User Interface) of Altova XMLSpy® 2005.

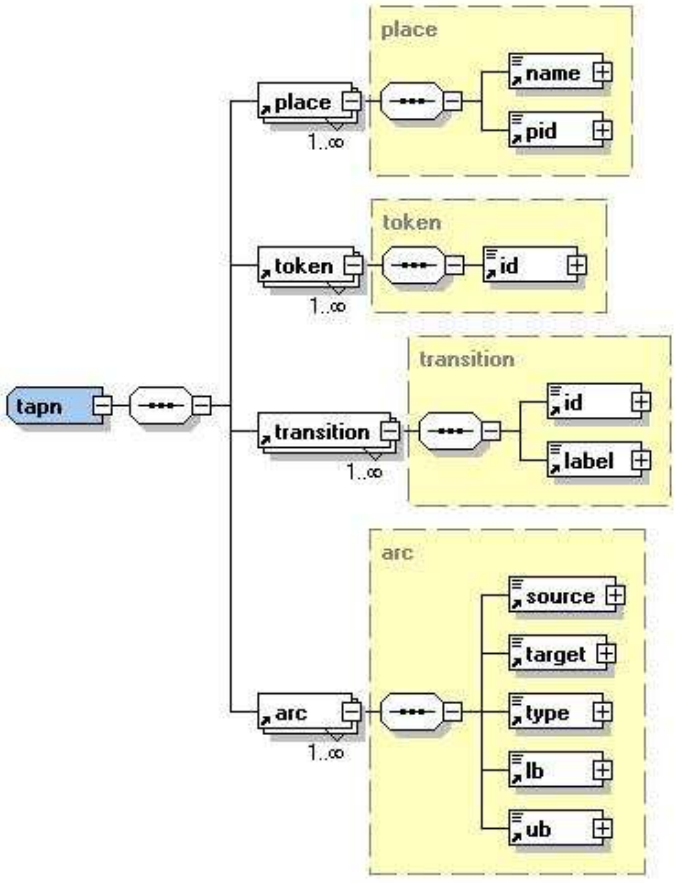


Figure B.1: DTD for the Input TAPN XML

B.2 DTD Code

```
1
2  <?xml version="1.0" encoding="UTF-8"?> <!--DTD generated by XMLSpy
3  v2005 rel. 3 U (http://www.altova.com)-->
4
5  <!ELEMENT tapn (place+, token+, transition+, arc+)>
6
7
8  <!ELEMENT place (name, pid)>
9      <!ELEMENT name (#PCDATA)>
10     <!ELEMENT pid (#PCDATA)>
11
12 <!ELEMENT transition (id, label)>
13     <!ELEMENT id (#PCDATA)>
14     <!ELEMENT label (#PCDATA)>
15
16 <!ELEMENT token (id)>
17     <!ELEMENT id (#PCDATA)>
18
19 <!ELEMENT arc (source, target, type, lb, ub)>
20     <!ELEMENT source (#PCDATA)>
21     <!ELEMENT target (#PCDATA)>
22     <!ELEMENT type (#PCDATA)>
23     <!ELEMENT lb (#PCDATA)>
24     <!ELEMENT ub (#PCDATA)>
```

C CD Packages

C.1 Readme

C.2 Tools

- **J2SE**
- **Uppaal**
- **Roméo**
- **TCL**
- **XMLSpy**
- **JDOM**

C.3 Program Packages

- **Howtos**
- **XML Parser**
  1. **Reader**
  2. **Writer**
     - **Reducing Degree**
     - **Translating into TA**

C.4 Examples

- **XML files**
- **Graphics**

Bibliography

[1] [AD90] Rajeev Alur, David L. Dill. *A Theory of Timed Automata. Theoretical Computer Science* 126:183-235, 1994. Preliminary versions appeared in *Automata, Languages, and Programming: Proceedings of the 17th ICALP*, LNCS 443, 1990, and *Real Time: Theory in Practice*, LNCS 600, 1991.

[2] [AL92a] M. Abadi and L. Lamport. *An old-fashioned recipe Report Systems Research Center 91*, Digital Equipment October 1992. To appear in *ACM Transactions on Programming Systems*. An earlier version appeared as [AL92b].

[3] [AL92b] M. Abadi and L. Lamport. *An old-fashioned recipe for real time*, 1992. In [7], pages 1-27.

[4] [AN01] Parosh Aziz Abdulla and Aletta Nyln. *Timed Petri nets and BQOs.* In *Proc. 22nd International Conference on application and theory of Petri nets (ICATPN)*, volume 2075 of LNCS, pages 53-70, 2001.

[5] [BD91] Bernard Berthomieu and Michel Diaz, *Member*, *IEEE. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. IEEE Transactions On Software Engineering*, vol. 17, no. 3, March 1991.

[6] [BDL04] Gerd Behrmann, Alexandre David, and Kim G. Larsen. *A Tutorial on Uppaal.* In proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04). LNCS 3185.

[7] [BHRR92] J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors. *Real-Time: Theory in Practice.* Number 600 in Lecture Notes in Computer Science. Springer-Verlag, 1992. Proceedings of the REX Workshop, Mook, The Netherlands, June 1991.

[8] [BSW69] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. *A Note On Reliable Full-Duplex Transmission Over Half-Duplex Link. Commun. ACM*, vol. 12, no. 5, May 1969.

[9] [BT87] Eike Best and P. S. Thiagarajan. *Some classes of Live and Save Petri Nets.* In K. Voss, H. J. Genrich, and G. Rozenberg, editors, *Advances in Petri Nets*, pages 71-97. Springer-Verlag, 1987.

[10] [CDK00] George Coulouris, Jean Dollimore and Tim Kindberg. *Distributed System: Concepts and Design, Third Edition (ISBN: 0-201-61918-0).* Addison-Wesley Publishers Limited 1984, 1988, and Pearson Education Limited 2001.

[11] [CEP95] Allan Cheng, Javier Esparza, Jens Palsberg. *Complexity Results for 1-safe Nets. Theoretical Computer Science*, 147(1-2):117-136, 1995.

[12] [CHEP71] Fred Commoner, Anatole W. Holt, S. Even, and Amir Pnueli. *Marked Directed Graphs. Journal of Computer and System Sciences*, 5:511-523, 1971.

[13] [CR04] Franck Cassez and Olivier-H. Roux. *From Time Petri Nets to Timed Automata*. IRCCyN/CNRS UMR 6597, BP 92101, 1 rue de la Noë 44321 Nantes Cedex 3 France, ©*Published by Elsevier Science B. V.* 2004.

[14] [DOM] `http://www.w3schools.com/dom/dom_intro.asp`

[15] [ERA+00] D. de Frutos Escrig, V. Valero Ruiz, and O. Marroquín Alonso. *Decidability of Properties of Timed-Arc Petri Nets.*

[16] [Fi85] M. Fischer. *Re: where are you?* Electronic mail message from Michael Fischer to Leslie Lamport. Arpanet message sent on June 25, 1985 18:56:29 EDT, number `8506252257.AA07636@yale-bulldog.yale.arpa` (47 lines), 1985.

[17] [GL04] Krishna Prasad Gundam and Vasu Hossaholal Lingegowda. *Model Checking Task Graph Scheduling with Timed Automata using* UPPAAL. SSE3 paper, Department of Computer Science, Aalborg University, 2004.

[18] [GL73] Hartmann J. Genrich and Kurt Lautenbach. *Synchronisationsgraphen. Acta Informatica*, 2:143-161, 1973.

[19] [GRM97] Michel Goossens, Sebastian Rahtz and Frank Mittelbach. *The LaTeX Graphics Companion.* Addison-Wesley, Reading, Massachusetts, 1997, ISBN 0-201-85469-4.

[20] [Ha72] Michel Hack. *The Recursive Equivalence of the Reachability Problem and the Liveness Problem for Petri nets and Vector Addition System.* In *Proc. 15th Annual Symposium on Switching and Automata Theory*, pages 156-164, 1974.

[21] [HS03] K. Van Hee, N. Sidorova. *Process Modelling and Analysis.* `http://wwwis.win.tue.nl/2M320/slides2003.pdf`, 2003.

[22] [JDOM] `http://www.jdom.org/`

[23] [JPXZ94] W. Janssen, M. Poel, Q. Xu, and J. Zwiers. *Layering of real-time distributed processes.* To appear in the proceedings of the *Third International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems*, Lübeck, Germany, September 1994.

[24] [KRONOS] `http://www-verimag.imag.fr/TEMPORISE/kronos/`

[25] [Lam87] L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems.* 5(1):1-11, 1987.

[26] [Li76] Richard J. Lipton. *The Reachability Problem Requires Exponential Space.* Technical Report 62, Yale University, 1976.

[27] [Mi89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989

[28] [NSS+00] Mogens Nielsen, Vladimiro Sassone, Jiří Srba. *Towards a Notion of Distributed Time for Petri Nets*.

[29] [OSJ05] Ragnhildur Óskarsdóttir, Sigmar Stefánsson and Tómas Jónasson. *On Deciding Behavioral Properties for Petri Nets: Timed-Arc Petri Nets and their Extensions*. Master's thesis, Department of Computer Science, Aalborg University, Denmark, May 2005.

[30] [Ou94] John K. Ousterhout. *TCL and the TK Tool Kit*. Addison-Wesley, Reading, Massachusetts, 1994, ISBN 0-201-63337-X.

[31] [Pe81] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Inc. 1981

[32] [Roméo] `http://www.irccyn.ec-nantes.fr/irccyn/d/en/equipes/TempsReel/logs/software-2-romeo`

[33] [SAX] `http://www.saxproject.org`

[34] [SBM92] F. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. In [7], pages 618-639, 1992.

[35] [Si97] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997, ISBN 7-111-10840-X.

[36] [Tina] `http://www.laas.fr/tina/`

[37] [UPPAAL] `http://www.uppaal.com`

[38] [Ve94] Jan Joris Vereijken. *Fischer's Protocol in Timed Process Algebra*. Department of Computing Science, Eindhoven University of Technology. August 18, 1994.

[39] [W3DTD] `http://www.w3schools.com/dtd/default.asp`

[40] [Wiki] `http://en.wikipedia.org/wiki/Main_Page`

[41] [WPD] Wang Yi, Paul Pettersson and Mats Daniels. *Automatic Verification of Real-Time Communicating Systems by Constraint-Solving*. Department of Computer Systems, Uppsala University, Box 325, S751 05, Uppsala, Sweden. Email:`{yi,paupet,matsd}@docs.uu.se`

[42] [XMLSpy] `http://www.altova.com/products_ide.html`