An incremental approach for the extraction of firing sequences in Timed Petri Nets : application to the reconfiguration of flexible manufacturing systems

Yongliang Huang

N° d'ordre: | 2 | 2 | 8 |

ÉCOLE CENTRALE DE LILLE

THÈSE

présentée en vue d'obtenir le grade de

DOCTEUR

en

Spécialité : Automatique Génie Informatique, Traitement du Signal et Images

par

Yongliang HUANG

DOCTORAT DELIVRE PAR L'ECOLE CENTRALE DE LILLE

Titre de la thèse :

Une approche incrémentale pour l'extraction de séquences de franchissement dans un Réseau de Petri Temporisé : application à la reconfiguration des systèmes de production flexibles

An incremental approach for the extraction of firing sequences in Timed Petri Nets: application to the reconfiguration of flexible manufacturing systems

---

Soutenue le 25 novembre 2013 devant le jury d'examen:

Président du Jury	Pr. Pierre MARQUIS	Université d'Artois
Rapporteur	Pr. Pascal BERRUET	Université de Bretagne Sud
Rapporteur	DR. Jean-Jacques LOISEAU	CNRS-IRCCyN
Directeur	Pr. Armand TOGUYENI	École Centrale de Lille
Co-encadrant	MCF Thomas BOURDEAUD'HUY	École Centrale de Lille
Co-directeur	MCF Pierre-Alain YVARS	École Supméca, Saint Ouen

Thèse préparée dans le Laboratoire d'Automatique, Génie Informatique et Signal
L.A.G.I.S., CNRS UMR 8219 - École Centrale de Lille
École Doctorale Sciences pour l'ingénieur ED 072
PRES Université Lille Nord-de-France

À mes parents,

à toute ma famille,

à mes professeurs,

et à mes chèr(e)s ami(e)s.

Acknowledgements

This thesis would not have been done by the help and supports of so many people in so many ways. I would like to take the opportunity to express my gratitude to thank all friends and colleagues that have given me support and encouragement during my thesis work. First of all, my sincere thanks go to all my supervisors, Prof. Armand TOGUYENI, Dr. Thomas BOURDEAUD'HUY and Dr. Pierre-Alain YVARS, for their valuable guidance, continuous patience and encouragement and the share of their research experiences.

I also would like to express my sincere gratitude to all the members in teams of "Systèmes Tolérants aux Fautes (STF)" and "Optimisation des Systèmes Logistiques (OSL)", it has been a privilege to work together with the intelligent and friendly colleagues. I would especially like to mention colleagues and friends in my office, Baisi LIU, Zhi Li, Rahma LAHYANI, Ramla SADDEM and Yue YU. Thanks to them, I have passed three agreeable years.

I would like to thank all the support staff. Throughout years they have been very helpful. I first thank of Prof. Philippe Vanheeghe, chef of LAGIS. The secretary has been run with perfectionism, Virginie Leclercq, Vanessa Fleury, Christine Yvoz and Brigitte Foncez have relieved me of the administrational work. My thanks also go to Bernard, Hilaire, Gilles, Jacques, Patrick and Regine for their help, humor and hospitality.

I wish to express my special appreciation to Prof. Héléne Catsiapis, my French teacher. In her courses, I have learned not only the language but also the culture and history.

I would like to thank all my friends in Lille for their friendship and supports.

Finally, I want to thank all my family for always supporting and encouraging me. Especially for my girl friend Jing HE, thanks for her waiting and supporting.

Contents

Chapter 1

Introduction

This thesis is the fruit of a three years work (2010-2013) spent in teams "Systèmes Tolérants aux Fautes (STF)" and "Optimisation des Systèmes Logistiques (OSL)" of "Laboratoire d'Automatique, Génie Informatique et Signal" (LAGIS) in Ecole Centrale de Lille. Under the supervision of Prof Armand TOGUYENI, Maître de conférences Thomas BOURDEAUD'HUY and Maître de conférences Pierre-Alain YVARS, I worked on extracting firing sequences in Timed Petri Nets (TPNs) with an incremental approach. Our approach can be addressed to the reconfiguration of flexible manufacturing systems

In this chapter, we introduce our motivation and the main problems in this thesis and relevant state of art.

1.1 Motivation

In the industrialized world, automation is taking a more and more important place. Machines, control systems and information technologies are used to optimize productivity in the production of goods and delivery of services automatically. A significant part of the "activity" in these systems is characterized by asynchronous occurrences of discrete events (like turning a machine "on"). Especially with the computer-based techniques developed in the information age, the "activity" lead systems to handle discrete states. These systems are thus called "discrete-state event-driven systems" (DES) [CL08].

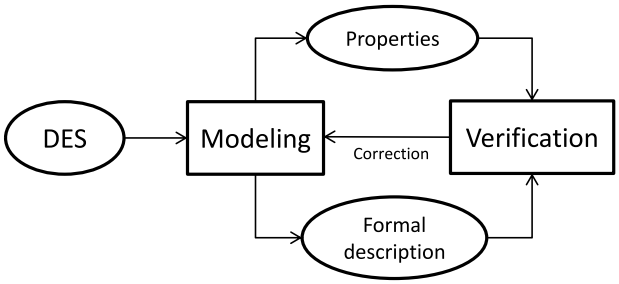


Figure 1.1: A Process of DES Modeling

In [Ver01], figure (1.1) is proposed to express the normal process of modeling DES. *Verification* is the necessary process to correct the *Modeling* based on formal descriptions and properties of DES. The most popular method of *Verification* is *Model Checking* [CGP99].

In recent years, increasing demands on *reliability* and *safety* of DES have met the need for extensive research for formal modeling and verification, which are taken a more and more important place. As an example, finding *all* possible faults and knowing when they happen are becoming very important tasks, which can be used for diagnosis. In real world manufacturing systems, the start time and duration of all faults (critical scenarios) will influence other events and the security of the production system.

Such needs lead us to find a powerful formalism for modeling and verifying *time critical* systems. However, model checking is not enough for giving details about actions leading models to faults or operations and their executing times to produce products in manufacturing systems.

In this thesis, we propose to generate *firing sequences in TPNs*, which can firstly tell the existence about faults (or critical scenarios), then generate *timed firing sequences* to give the information of timed actions or events leading to them. Thus, the main work in this thesis can be divided into two steps:

- Solve the ***reachability problem*** of TPNs.
- Give timed ***firing sequences*** leading to a desired marking.

Since finding firing sequences can be also a verification process for the time-critical model, we describe the main process of modeling time-critical DES by

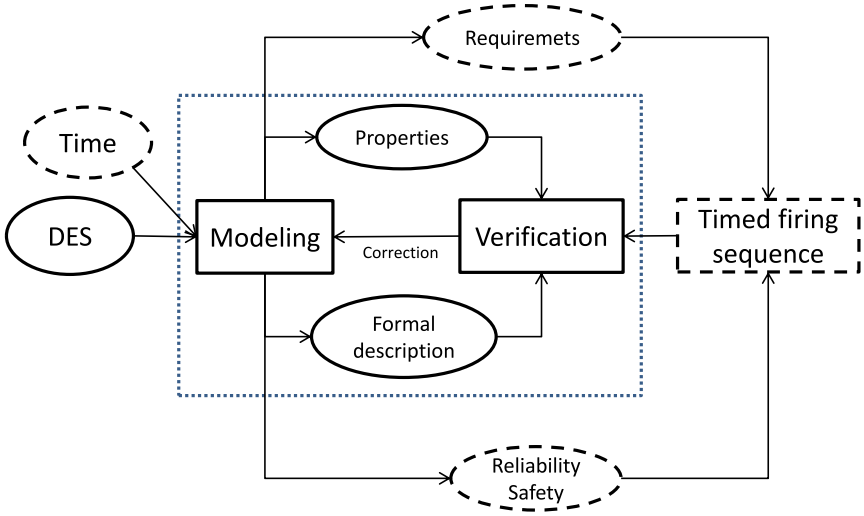


Figure 1.2: Generating Firing Sequences for DES

figure (1.2). Note that *Requirements* represent needs for applying models to accomplish their functions.

However, for generating firing sequences, since it will take much effort on *recording the information* from initial state to the desired state, only a few effective methods have been proposed.

Especially, since generating *all* possible firing sequences is an important verification process, we propose to use an enumerative approach – *Constraint Programming* (CP). Meanwhile, combining TPNs and CP technique leads us to model systems and find solutions more efficiently.

1.1.1 Petri Net Formalism

For modeling and analyzing DES, there are several formalisms, like probabilistic modeling, exotic algebras, automata, grafcet, state charts, Petri Nets (PNs), etc. All these techniques can be adapted to the characteristics of considered systems and specific needs of users. The interested reader should consult with profit the introductory part of the course of [HZ07] and [CL08] for a discussion on these tools.

Among them, technical description and formal verification provide a mathematical framework to rigorously validate all stages of design. [MV98] indicates that the most common approach for the formal description technique validation is to associate them a graph of reachable states. Nodes of the graph represent states of the system, its evolution arcs or transitions. Modeling a state graph is exhaustive: all states and all possible transitions of the system must be present in the graph.

Automata and *PNs* have been studied as powerful approaches for modeling and controlling DES. They can both be associated with a state graph. In automata, this is done by explicitly enumerating all possible states and then "connecting" these states with possible transitions between them, resulting in the transition function of the automaton. This is not particularly graceful, yet, automata are easily combined by operations such as product and parallel composition and therefore a model of complex system can be built from models of individual components in a systematic manner. PNs on the other hand have more structure in their representation of the transition function. States are not enumerated; rather, state information is "distributed" among a set of places that capture key conditions that govern the operations of the system. In the general case, an automaton can always be represented as a Petri Net (PN); on the other hand, not all PNs can be represented as finite-state automata. Consequently, PNs can represent a larger class of systems [CL08]. In addition, PNs are applicable to a wide variety of DES because of its excellence in analytical ability and the operation ability [MKI07].

Therefore we choose PNs as the main formalism of this manuscript.

However, "untimed" PNs are not powerful enough to deal with performance evaluation, safety determination, or analysis of behavioral properties in systems where time appears as a quantifiable and continuous parameter. For instance, when the assumption that all transitions can fire instantaneously is no longer a good approximation.

Such consideration leads us to focus on generating firing sequences in TPNs.

1.1.2 Generating Firing Sequences for Timed Petri Nets

PNs were first introduced by Carl Adam Petri in [Pet62]. Since their creation, PNs evolved from a sound representation of discrete dynamic systems into a general schema, able to represent knowledge about processes and (discrete and distributed) systems according to their internal relations [SF12].

There are many extensions of PN models [HZ07]. According to [DA94], they can be classified as abbreviations or extensions. Abbreviations of the basic PN model enable one to represent a DES in a simplified way, for example, colored PNs [Jen92], etc. On the other hand, extensions are PN models with additional functional rules to those defined for the basic model like deterministic, stochastic, timed [Wan98], etc.

Among several proposed extensions to deal with time, we introduce two basic models: Ranchamdani's Timed Petri nets [Ram73] and Merlin's Time Petri nets [MF76]. Since time inscriptions in these two temporal PN models are always associated to *transitions*, they are so-called T-time PNs. Other time extensions have been published including some approaches where time is associated to places or arcs, which are called P-time or A-time PNs. (see [CMS99] for a survey).

Many control problems in DES can be modeled as *reachability problems* in PNs, such as flexible manufacturing and communication systems [MKMH86]. The reachability problem can be solved by searching for the existence of a non-negative integer solution of the matrix equation of the net. Even if one exists, however, there is still the problem of finding an appropriate firing sequence, which also takes exponential time and spaces [XZ98].

Reachability problem can be seen as a subproblem of generating firing sequences. Firing sequences [MKI07] are quite fundamental in the sense that find-

ing a firing sequence of transitions from an initial marking (initial state) to the target marking (state of the target), is a sub-problem of various basic problems of PN theory. Many manufacturing problems can be solved by searching for firing sequences, like classical scheduling problems, the minimum initial resource allocation problem and the well-known marking reachability problem, asking for a firing sequence whose firing leads from a given initial marking to a specified target marking, see a reference in [YW94].

The problem to determine a firing sequence from initial marking to a reachable target marking is known to be NP-hard. The exact algorithms to solve a firing sequence problem have been studied in literature [YW94]. For example, the heuristic search, hybrid heuristics with backtrack, and genetic algorithm have been reported in past works for scheduling problem by using TPNs. Conventional optimization models for PNs are concentrated on its search in the entire system.

However, these approaches cannot be applied for large scale systems due to the well known problem of combinatorial explosion.

1.2 The Problematic

1.2.1 The Problem of Combinatorial Explosion

The reachability graph of PNs is very convenient for behavior analyzing and verification technique in concurrent systems [Val98]. However, it suffers from the combinatorial *state* explosion problem. As known, PNs have a compact representation for concurrent systems, but when it comes to check properties (like critical scenarios) in system, one has to search through the underlying reachability graph which reduce all the benefits of the compact representation.

The example of figure (1.3), which is derived from ([LNC+08]) describes a family of manufacturing systems characterized by three parameters: n, m and k.

- n is the number of production lines.
- m is the number of units of the final product that can be simultaneously produced. Each product is composed of n parts.
- k is the number of operations that each part must undergo in each line.

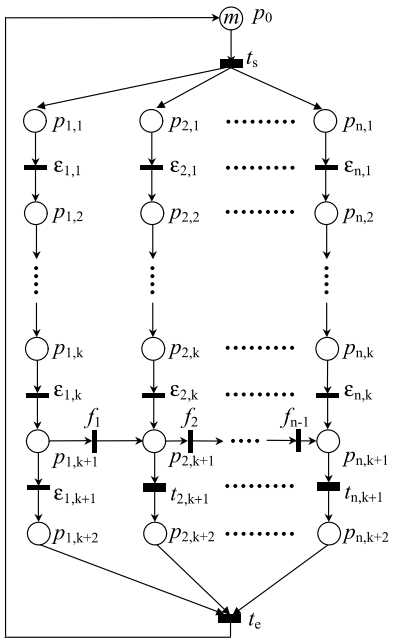


Figure 1.3: An Example of Combinatorial Explosion

Using the PN tool *TINA* [BRV04], numbers of states in the reachability graph of figure (1.3) are given in table (1.1). As been shown, the number of states in the reachability graph will grow exponentially, especially when the number of tokens m grows.

m	n	k	$\|\mathrm{R}\|$
1	2	1	15
1	2	4	48
1	4	1	495
3	4	1	484841

Table 1.1: Number of States in Reachability Graph of figure (1.3)

Therefore, many studies have focused on developing techniques for *state space* reductions based on the PN graph or its formal representation to capture the subset of states.

Before introducing methods for solving Time Petri Net (TPN) reachability problem, we propose to establish state of art briefly concerning main approaches to handle combinatorial explosion for the analysis of PNs. We are inspired

strongly by presentations of [Ben00] [Ver01] as well as several book chapters [Dea01, Dea03].

In this chapter, a simple example and its reachability graph are presented in figure (1.4), which is taken from figure (1.3), to illustrate methods in this chapter. Here we make $m = 1, n = 2, k = 2$.

Note that states $00200, 00002, 00101$ are faults.

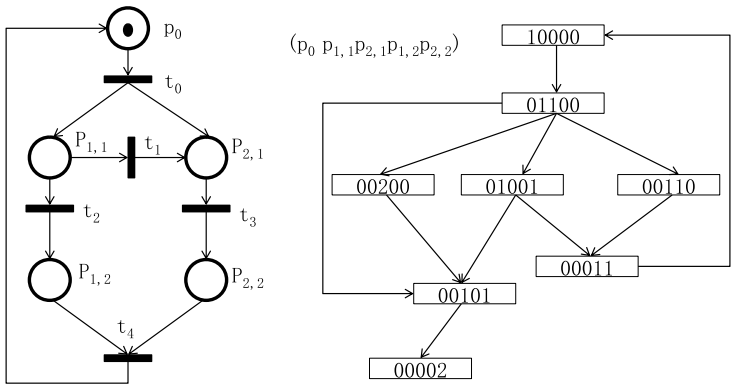


Figure 1.4: An Simple Example and Its Reachability Graph

1.2.1.1 A Representative Problem: Reachability

It is precisely in this context that the problem of combinatorial explosion arises, since the size of the graph can evolve exponentially with the size of the system studied.

For example, consider the reachability problem in PNs, which is to decide if a given marking is reachable from the initial marking. This problem is known to be representative of the limitations imposed by combinatorial explosion. Even if we can show that it is decidable (see the proof of [Kos82]), it is *EXP-TIME* and *EXP-SPACE* in the general case [Lip76].

Methods based on the construction of the underlying reachability graph become impassable if the size increasing quickly. In fact, we can show that even in the case of *bounded* PNs - a special class that the reachability graph is finite - the problem of constructing a reachability graph is not primitive recursive, which is to say that "any method based on the construction of the reachability graph has

a complexity that cannot be predicted" as Haddad said in chapter 4 of the book [Dea01].

However, most problems of PN analysis (including issues related to the liveness and deadlock, which are taking great interest in the verification process) are related to the problem of reachability, as indicated by [AK77], and are subject to the same limitations. Many works are made to control this combinatorial explosion.

[Ver01] proposes to classify these techniques to limit the combinatorial explosion into two categories: state space management and state space reduction.

State space management uses sophisticated techniques to present state space more efficiently. State space reduction techniques take advantage of disciplines, redundancies or similarities in the comprehensive representation to reduce the complexity of the system.

It should be noted that these approaches are complementary: they can be applied together to win efficiency.

1.2.1.2 State Space Mangement

In this category, we classify methods as graph compression and verifying on the fly.

Graph Compression

The goal of graph compression is to present the original graph with a more *understandable* way without losing its characteristics. The most famous form of compression technique is *Binary Decision Diagram* (BDD), which is first introduced by [Ake78] to encode concisely boolean functions of several variables.

A BDD is a directed acyclic graph. In popular meaning, the term BDD almost refers to Reduced Ordered Binary Decision Diagram, which comprises a root and two terminal nodes (0 and 1). Each intermediate node is labeled by a boolean variable and has two outgoing edges labeled by 0 and 1.

The BDD represents a boolean function as follows: each assignment of all variables corresponds to one path in the graph from the root node to one of the two terminal nodes.

Several extensions are developed based on BDD, for example Multi-Valued Decision Diagram (MDD), Algebric Decision Diagram (ADD), etc. For MDD, it just means that each node can have more outgoing edges based on the variables' domain.

To illustrate the BDD or MDD, we consider the example in figure (1.4) and give its MDD as shown in figure (1.5). **Note that the two terminal nodes 0 are the same one, which are used to make this graph more clear. Meanwhile, if there were no values 2, it will be a BDD.**

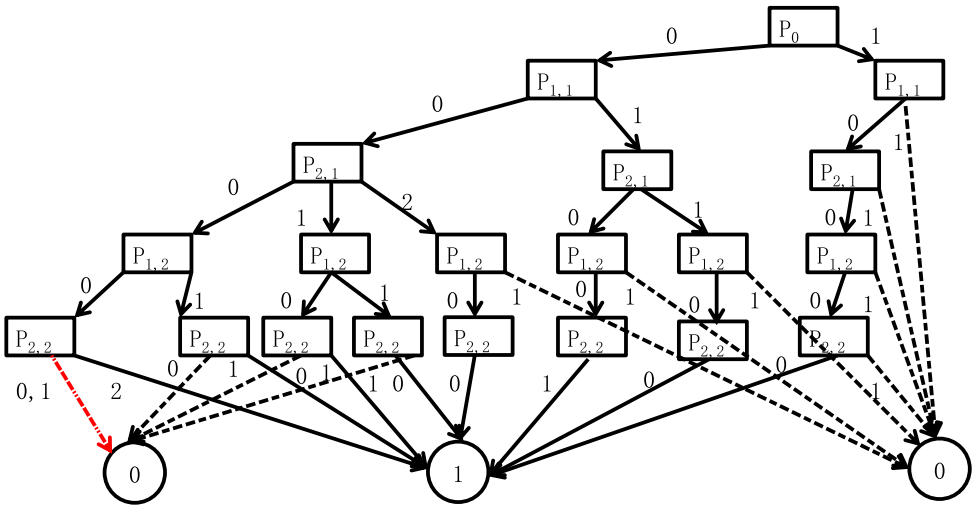


Figure 1.5: The MDD for Figure (1.4)

Note that the *complete* ***search space*** is divided into *reachable states* and *impossible states*, which are represented by paths leading to the terminal node 1 and 0 respectively. Therefore, BDD/MDD exactly encode the behavior graph of PNs, it is theoretically possible to check all properties of a system based on BDD/MDD representations.

However, since the BDD/MDD is not an *abstraction* graph, they will be infinite if the initial graph is infinite. The biggest problem is to control the compression rate. In fact, it strongly depends on variables' order. For that, [BCM+90] propose to use sophisticated scheduling techniques variables.

In addition, transitions between reachable states are not presented as in the reachability graph, which means they do not supply the firing sequences.

On The Fly Verification

Generally, system verification takes place in two stages. The reachability graph is constructed in the first stage, then decision algorithms are executed on it at the second stage.

The *on the fly* technique proposes to perform these two tasks in parallel. Thus, satisfied properties are evaluated during enumeration of behavior of PNs.

We can still take figure (1.4) as an example. If faults are needed to be detected, the *on the fly* technique can label variables like in figure (1.5).

If the most general *depth-first* algorithm is applied for exploring the *search tree*, the fault state 00002 will be found after meeting two impossible ends 00000 and 00001, which are presented with the same red dotted line in figure (1.5).

The advantage of the *on the fly* technique is apparent. Since only states on the path need to be stored, the memory required is lower than methods that firstly build the whole reachability graph.

However, if the desired state is situated at the last part of the search tree, most parts of the search tree are still explored. This means that the *on the fly* technique does not make reduction of the reachability graph.

The interested reader can refer to the article [FJJM92] for more details on these techniques.

1.2.1.3 State Space Reduction

The common point of approaches presented in this subsection is that they restrict model by defining an incomplete behavioral model, which will be smaller or less accurate than the original one. Then verification on this reduced model will be more effective or at least feasible.

These proposed mechanisms are generally specific to the desired property. They only allow to treat a subclass of properties that are known to be preserved by the restriction mechanism. Thus, it may be necessary to repeat the reduction process, which is also expensive, for checking each class of properties.

These restriction operations can be made based on properties that will be verified and specifications that will be handled in the system. We distinguish these techniques as: reductions behavior graph, transformations and decompositions of

PNs and directly studying equations associated with PNs without changing its structure.

Algebraic Methods

Algebraic methods [Lau87] use the *state equation* to analysis the behavior of PNs. They often use linear algebraic methods to deduce the structural properties. We can refer to examples in [LM89] and [STC98].

The verification of certain properties can be reduced to a linear programming problem using the *duality* theory [CPJ80]. Analysis techniques based on linear algebra allow the verification of properties of a general net system. The key idea is simple: Let $\mathbb{S}$ be a PN with the incidence matrix C. If m is reachable from m_0 by firing sequence σ, then

$$m = m_0 + C \cdot \sigma \tag{1.1}$$

Therefore, the set of natural solutions (m, σ) of this state equation defines a linearization of the reachability set. This set can be used to analyse properties such as marking and submarking reachability and coverability, etc. To do so, the properties are expressed as formulas of a first order logic having linear inequalities as atoms, where the reachability or fireability conditions are represented by the state equation. These formulas are verified by checking the existence of solutions to systems of linear equations that are automatically obtained from the formulas.

Thus, the reachable markings are compressed as linear equations by the state equation. The *boundedness*, *liveness* and *safety* properties of PNs can be analyzed easily using linear algebraic methods.

However, linear algebraic methods just give an *approximation* reachable marking set E. When the set E increases larger, it becomes more difficult to obtain. This approximation is often insufficient. Indeed, methods based on state equation are semi-decision algorithms, since the state equation is only the necessary condition for a reachable marking. There may be false solutions (*spurious solutions*) that do not have a valid firing sequences.

Transformations and decompositions of Petri nets

To reduce the size of reachable markings set E, [Ber86] has developed techniques that can reduce the size of PN while preserving properties: liveness, blocking, etc as appropriate.

These techniques involve applying transformation rules or decomposition. A transformation is to change a PN. This change applies to places or to transitions that are merged. Decomposition is to consider sub-nets separately and then to deduce the behavior of the entire net.

The simplest transformation is to remove redundant places and arcs connected to it. The presence or absence of these kind of places do not affect the possible behaviors of the net. It is also possible to merge places under certain conditions. Finally, transitions will be merged when all firing sequences can be rearranged so that some transitions are always paired.

The principle of decompositions involve analyzing a set of smaller net to extract properties. Then to analyze the entire network by composing them. The composition is done by identifying the places or transitions of sub-nets. The problem is that some properties are not necessarily preserved in the composed net.

Reduction of the Reachability Graph

Coverability graph. In the case of unbounded PN, the underlying behavior graph has an infinite number of markings. The coverability graph is defined by [KM69] as the first reduction technique associating to the infinite graph. Then, it is formalized more precisely by [Fin93]. This graph gives a finite representation of the partial set of reachable states.

When a system has an infinite set of states, some parts of markings are not bounded. We therefore introduce the notion of *pseudo-marking*, which is a mapping from the set of places P to $N \cup \omega$. Pseudo-markings are a generalization of markings in the sense that a pseudo-marking is a marking that can contain an infinite number of tokens. In fact, a pseudo-marking represents a class of markings which places are likely to contain an arbitrarily large number of tokens.

The principle of the coverability graph is to represent the set of reachable markings by a set of pseudo-markings. This set covers all reachable markings, which is the coverability graph. In addition, any pseudo-marking graph is the limit of a sequence of reachable markings: the coverability graph contains no useless nodes.

The main results on the coverability graph are:

- The coverability graph is finite;
- Given a PN and a marking m_0, we can decide whether the reachability graph contains a marking $m' \geqslant m_0$.

Since the coverability graph is finite, it is theoretically possible to build and use it to deduce properties. However, analysis methods using this graph are limited.

We now present techniques more "suitable" to reduce behavior graphs based on properties or to check the original behavior features.

Behavior Abstraction. From the view of properties that will be checked, some details of the studied system are unnecessary. Methods of *behavior abstraction* define a formal framework to consider simplified problems.

These techniques use the relationship between the initial graph and reduced graph, called *abstraction*. These relations must hold relations between transitions and places to a certain extent, so that properties of the reduced graph are constructed to be representative of properties of the initial graph. The interested reader can refer to the thesis [Loi94] for a complete bibliography on this subject.

Abstraction is one of the few ways to reduce infinite behavior graphs by finite graphs. In addition, results of conservation concern a broad class of properties to be checked.

However, the construction of the abstraction system slows down a wider diffusion of this method. Indeed, it is not automatic and necessary specific methods. Therefore, the research focuses currently on identifying general patterns of abstraction in order to automatize this type of approach.

Partial order. Partial order techniques (see [God96] for a complete explanation) have been developed to limit the combinatorial explosion by attacking one of its causes: the representation of parallelism by interleaving actions.

When two independent actions lead to the same global state, we say that they are interleaving in the reachability graph. These two actions are independent and therefore lead to the same overall condition.

Formally, two actions are said to be independent iff:

$$\forall p, p_a, p_b, \text{if } p \xrightarrow{a} p_a \wedge p \xrightarrow{b} p_b \Rightarrow \exists p' s.t. p_a \xrightarrow{b} p' \wedge p_b \xrightarrow{a} p' \tag{1.2}$$

Two independent actions are said to be conflict. It is possible to use this independence relationship to define an equivalence relation on the firing sequences of the behavior graph: two firing sequences are equivalent iff they can be obtained one from another by permuting adjacent independent actions.

The terminology *partial order* is that the equivalence class is a partial order on the occurrences of actions. The idea is to reduce the behavior graph by equivalence trace. There are basically two types of approaches to these methods giving different reduction relations.

- Interlacing elimination: seek a subgraph of the behavior graph with the least possible equivalent traces, for example *stubborn sets* [Val91].
- Covering step: gather sets of independent actions, introduced by [VAM96].

The partial order methods are very beneficial in two ways:

- the graphs obtained are often very small compared to the original graph
- The time required to calculate the relationship of independence is low compared to the complexity of the system

As a result, these techniques allow to save both time and memory required to generate the behavior graph. As for the BDD, the application to real protocols demonstrated the effectiveness of these techniques.

Symmetry. The systems studied often have a symmetrical architecture. The elimination of all symmetric and redundant situations in the behavior of the system allow to simplify the study. [HJJJ85] formalized the notion of symmetry to reduce the size of reachability graph. They introduce an equivalence relation, called symmetry between the nodes of the graph on the one hand, and the other

arcs. This gives rise to the construction of a reduced graph where each node represents an equivalence class of nodes of the reachability graph.

1.2.2 Timed Petri Net Reachability problem

Since time are needed to be handled between states, the problem of combinatorial explosion will be enlarged with the TPN.

Several approaches have been proposed to solve the TPN reachability problem, either by restricting their study to a subclass of Time PNs, like Timed Event Graphs, either by using dedicated heuristics. A complete bibliography can be found in [Ric00].

1.2.2.1 Subclass of Time Petri Nets

Since the fire of a timed transition can occur as soon as it is fireable and as late as one wants, there may exist, from a given state, an infinite number of reachable markings (depending on the time), and no reachability graph can be built. A first approach needs to consider TPN as a subclass of Time PN, in order to use the state enumeration methods (*state class graphs*) proposed by [BD91].

In [BD91], the state classes of time PNs are defined as a pair $S = (M, D)$:

- M is a marking, the marking of the class: all states in the class have the same marking;
- D is the firing domain of the class; it is defined as the union of the firing domains of all the states in the class.

All states with the same marking M are represented in the same state S. Since firing times are real numbers, one state S can represent infinite states of time PNs with the same marking M. Thus, the reachability graph of bounded time PNs can be represented with a finite *state classes*.

In fact, this method mainly concerns the expression of states of time PN, but not the verification of the reachability problem. It does not give much improvement on resolving reachability problem.

In general sense, if firing time domains D are not considered, the state class graph will be the reachability graph of corresponding PNs. Then one can constrain time to each state to find the feasible firing sequence.

More general solutions can be found in [DYKG04], where untimed firing sequences, which can directly leading to the desired marking, are generated first. Then firing times are associated to firings by propagating time constraints. Unfortunately, such method requires to enumerate all firing sequences first, which lead to combinatorial explosion, and it may exist an infinite number of associations of dates to firings (if a transition can be fired at date d, it is still fireable any moment later).

1.2.2.2 Controlled Execution

In [CC88], the *Controlled Execution* is proposed to represent the behavior of TPNs, which will be detailed in chapter (2). The main idea is to associate transitions with a successive firing dates to represent the behavior of TPNs. In this method, *early semantics* (a transition is fired as soon as it is fireable) is considered, which can be used to proceed to an enumerative and structural analysis [DA92], and to build the earliest state graph [CC88]. Note this method cannot handle the general timed reachability problem since firing transitions as soon as they are enabled may lead to miss some optimal firing sequences. For instance, let us consider the TPN of figure (1.6). The earliest firing sequence $([t_1t_4, 0], [t_5, 1], [t_2t_6, 11], [t_3, 12])$ needs 10 time units* to finish and takes 22 t.u. together. However, the optimal solution is given by the sequence $([t_1t_4, 0], [t_2, 2], [t_3t_5, 3], [t_6, 13])$ which needs 1 t.u. to finish and 14 t.u. in total.

1.2.2.3 Max-Plus Algebra

Max-plus algebra is a *dioid* algebras. The term dioid refers to the fact that this algebra is based on two operations. The operations are formally named *addition* and *multiplication* and denoted by $\bigoplus$ and $\bigotimes$ respectively. However, their meaning is different. For any two real numbers A and B, functions are as follow:

*we use t.u. for short in the following context

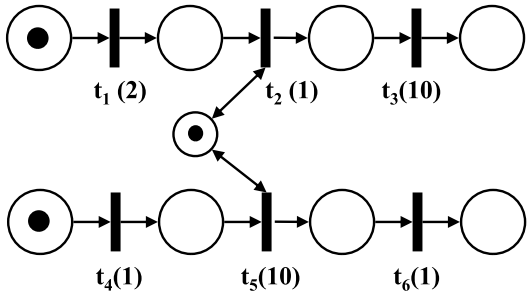


Figure 1.6: Earliest Firing Date Does not Mean Optimality.

$$\text{Addition:} A \bigoplus B \equiv max\{A, B\} \tag{1.3}$$

$$\text{Addition:} A \bigotimes B \equiv A + B \tag{1.4}$$

The time of transitions' firing and finish in TPNs can be addressed by these two algebras. The early semantics has been extensively studied for the special class of Timed Event Graphs (where a place has exactly one input and one output transition), using $(max, +)$ algebra [BCOQ92, ADL12]. Since their structure does not handle conflicts, it is possible to obtain linear equations corresponding to the complete behavior of the net.

1.2.3 Reconfiguration of Manufacturing Systems

In this thesis, we propose to generate firing sequences for TPN based on solving TPN reachability problem. The problem of reconfiguration of manufacturing system consists in searching for firing sequences in order to know the exact time of reconfiguration actions. Thus, a reconfigurable transport system is applied to valid our approaches in this thesis.

The concept of flexible manufacturing systems (FMS) was introduced in the eighties to develop new manufacturing production systems able to produce small and average series of products. But dependable requirements and especially the necessity to continue to produce despite of the breakdown of a plant component led to exploit the flexibility to *reconfigure* the plant.

Based on these consideration, the concept of *reconfigurable manufacturing systems* (RMS) has been introduced in the last of nineties by [Ber98, KHJ+99, AL04]. A RMS must be able to adapt its configuration in real-time depending on production objectives and available resources.

The reconfiguration process [Ber98] is a control function that aims to reorganize the structure and the software of an *automated production system* in order to satisfy the user requirements with regards to production, and to minimize the number of resources in production.

The reconfiguration process consists in two stages [TBC03, DTDC00]: the *decision-making* part and the *operational process*. The decision-making part consists in determining for a plant, a new *objective state* to reach. The operational process consists to determine the *procedure to apply* to reach this objective state from the current faulty state. To deal with these two stages, several propositions are currently being developed.

Concerning the *decision-making* part, many works have been done in France. In the LAB-LAGIS, authors applied Artificial Intelligence techniques to decision-making part [Tog04, BT09]. In the LAB-STICC, [DLBP05] has developed some criteria, which allow off-line and on-line analyses of the flexibility and of the reconfigurability of a plant.

Concerning the *operational process*, a lot of researches are on the way to apply reconfiguration procedures in order to obtain a control that is coherent with the *real structure* of the operative part. As an example, at the LAB-GIPSA (Grenoble, France), scheduling techniques are proposed for the *on-line synthesis* of a controller that respects the constraints of the operative part even in response to a failure [HDZJ04]. At the CRAN (Nancy, France), the proposed approach is based on a holonic control of the plant [GPG04]. This approach is characterized by a coupling of *product controllers* and *resource controllers*. Product controllers manage the routing of parts in the plant according to their transformation sequences (user specification) and depending on resource capabilities. Resource controllers manage the execution of the operations requested by the product controllers.

In this thesis, we develop a TPN model for analyzing reconfiguration of manufacturing systems. Using TPN models, firing sequences can express the reconfigure actions and their occurrence date.

1.3 Main Methods in this Thesis

As said in the motivation, the main work in this thesis can be divided into two steps:

- Solve the *reachability problem* of TPNs. The main difficulty is to reduce the influence of combinatorial explosion.
- Give timed *firing sequences* that leading to the desired marking. The main difficulty is to record information about the firing sequence.

Since the reachability problem is the basis for generating firing sequences, we have introduced many methods to solve the reachability problem in TPNs. However, firing sequences or the reachability problem cannot be easily solved, when large scale system or interleaving behavior are needed.

Therefore popular methods will search for the desired marking using *on the fly*, *reduction graph* or *behavior abstraction*, etc. These methods have to resolve under the whole ***search space*** of a TPN model, which contains reachable states and impossible states like presented in the figure (1.5). Reachable states form the underlying reachability graph of the TPN.

In general, a ***search tree*** should be explored to find the desired marking, as shown in figure (1.7). Since the desired marking at the i depth of the search tree, branches beyond the depth of $i + 1$ are not neccessary to be explored.

Thus, in this thesis, we firstly propose to follow the idea of **bounded model checking** (BMC) [BCCZ99] to explore the search tree incrementally, which can lead the search into a finite graph, even if the original graph is infinite.

1.3.1 Incremental Approach

The reachability problem of PNs can be seen as a verification process. Techniques for automatic formal verification of finite state transition systems have been developed since many years. The most widely used method is called *Model Checking* [Cla08], in which BMC has give us the main idea of searching for solutions *incrementally*.

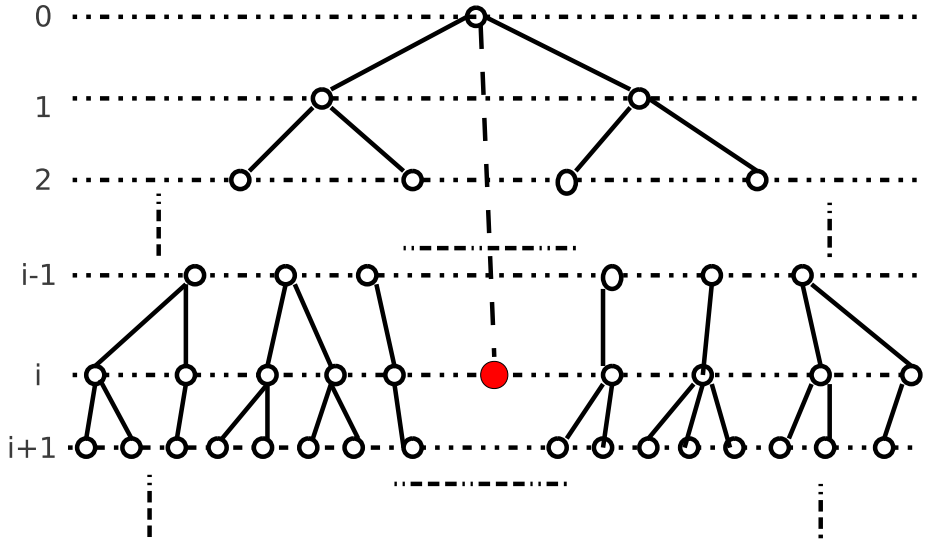


Figure 1.7: A Search Tree with One Desired Node.

In model checking, the specification is formalized by writing *temporal logic* properties. The system to be verified is modeled as state-transition system. Then, temporal logic properties can be verified based on this state-transition model. This state-transition model can be represented by a *Kripke structure* M, which represents set of states and corresponding transition relation, etc. *Propositional Linear Temporal Logic* (PLTL or LTL for short) is applied to express properties as temporal logic formula f. Then model checking can be summarized as an algorithmic technique for automatically checking temporal properties of finite systems [BCC+03].

The process of verifying properties is similar to search for reachable marking in the reachability graph of PNs. They both have the problem of combinatorial explosion. Therefore *Symbolic Model Checking* [BCM+92] was first introduced to reduce the influence of combinatorial explosion, combined with the BDD. In symbolic model checking, sets of states are represented implicitly using boolean functions and represented by BDD. Then the procedure starts an iterative process to verify the temporal logic properties until an error or a counterexample is found.

The bottleneck of symbolic model checking with BDD is the amount of mem-

ory needed for storing and manipulating BDDs, since the BDD never reduce any states in the underlying reachability graph, which will suffer from the problem of combinatorial explosion.

BMC was first proposed by [BCCZ99]. LTL formula is given in *negation normal form* to represent properties that will be verified. This consideration will lead solver to search for a counterexample instead of verifying some properties to be *true* in the whole sets of states. Then the BMC will be reduced to a propositional satisfiability problem (SAT problem) and solved by SAT methods.

A propositional formula $[\![M, f]\!]_k$ is given based on the LTL formula f and the *Kripke structure* M. As been said, this formula $[\![M, f]\!]_k$ will be resolved using SAT solver. Under a bound number k, SAT solver will search for all possible paths to find the counterexample. If no such counterexample has been found, k will be increased until reached the upper bound (this bound is called *Completeness Threshold*, marked as K_{bmc}).

Take figure (1.7) as an example, since the depth of the desired marking (the red node) is not known at the beginning, solver will have to search from the depth $k = 1$. Then solver will search and record all reachable states at each search depth in order to explore the search tree in the next search depth, until the red node reached.

The BMC does not suffer from the space explosion problem as BDD-based methods. However, it still has to search for all states of all paths if no counterexample can be found under a bound k. Meanwhile, BMC just gives the answer that if properties are satisfied or not. However, firing sequences leading to the desired marking are needed to show the exact actions in the real-time systems.

Thus, in this thesis, we secondly propose to adapt the *Logical Abstraction Technique*, introduced in [Ben00], [Yim00] to solve the reachability problem in TPNs. This technique is based on an *implicit traversal* the reachability graph of PNs, without its whole construction. The key idea is to use a *partial steps* to express the behavior of PNs, which can.

1.3.2 Logical Abstraction Techniques

The logical abstraction technique has been proposed in the thesis [Ben00] to solve the reachability problem of PNs. It is used to search for firing sequences leading to the final desired marking using a constraint solver.

The formalization described in this work is based on the concepts of *steps* and *sequences of steps* that we also use later in this document (see in subsection (2.2.2)). Informally, a step is simultaneous firing a set of transitions (concluding the reentrance of the same transition), which has also been introduced in the thesis [Bou04].

Benasser defines objects called *"partial" markings* and *"partial" steps*. Informally, it is to consider steps and intermediate markings as vectors of *variables* which are associated with formulas. Formulas corresponded to *constraints* on variables ensure that all possible instantiations of these formulas still represent markings and concrete valid steps.

Benasser shows a sequence, called *complete* partial steps with sequences of length k, captures exactly all stretchable sequences within this length of steps. More specifically:

- Any *sequence* of k steps corresponds to a particular instantiation of a complete sequence of a k partial steps;
- All possible *instantiations* of a complete sequence of partial steps correspond to a sequence of concrete valid steps.

On the other hand, from the viewpoint of reachable markings, the partial markings produced by the sequence of *complete* partial steps are all reachable markings within k steps. Therefore, it is the same to build the reachability graph after k steps and to build the complete sequence of *partial steps* of length k.

Take figure (1.7) as an example, each depth i of the search tree will be represented as a step σ_i. Then all reachable states within the depth i can be reached by the sequence of $\sigma_1, \ldots, \sigma_i$.

In this thesis, we propose to adapt the logical abstraction technique to TPNs, so that the behavior of TPNs can be expressed by a *sequence* of timed step. Note

that a *timed* step can fire a set of transitions, but cannot fire reentrance for one transition. In other words, transitions in TPNs are *mono server*.

In a bounded TPNs, all reachable marking of the underlying reachability graph can be expressed by the sequence of timed steps under a bounded integer number k. However, the whole underlying reachability graph of TPN is not needed to be explored for solving a specific reachability problem, which is situated in the front part of the search tree.

Therefore, we propose to adapt advantages of Logical Abstraction of PNs and BMC to solve TPN reachability problem.

1.3.3 Incremental Approach for Solving TPN Reachability Problem

To illustrate our method, one example is given in figure (1.8). The motivation, which is to find a firing sequence in TPNs, can be translated as finding a path leading from the root node to the desired marking and giving all values of all nodes belonging to the path.

The idea of BMC is used to search for solutions incrementally, avoiding to explore the whole size of search tree. Since the desired marking is situated at the $i+2$ depth of the search tree, BMC will search from depth 1 to $i+2$ to find this solution.

Since BMC has to search for all states and all paths before $i+2$ depth, which will cost too much memory to record previous states, we propose to use the logical abstraction technique to an implicit traversal the search tree.

In figure (1.8), each depth i of the search tree is represented by a timed step ψ_i. When the search depth is smaller than $i+2$, the method of *timed step* still have to search the whole search tree of length $i+2$ to verify that there is no solution. However, the method of timed step will avoid to search for some branches that cannot participate in a solution based on constraint relations. These kind of branches are called *deadends*, which are with black color in figure (1.8).

Note that if the search depth $i+2$ is *first* given to search for solutions, many branches are not needed to be explored. In figure (1.8), if depth-first algorithm is applied, paths after the solution are not needed to be explored.

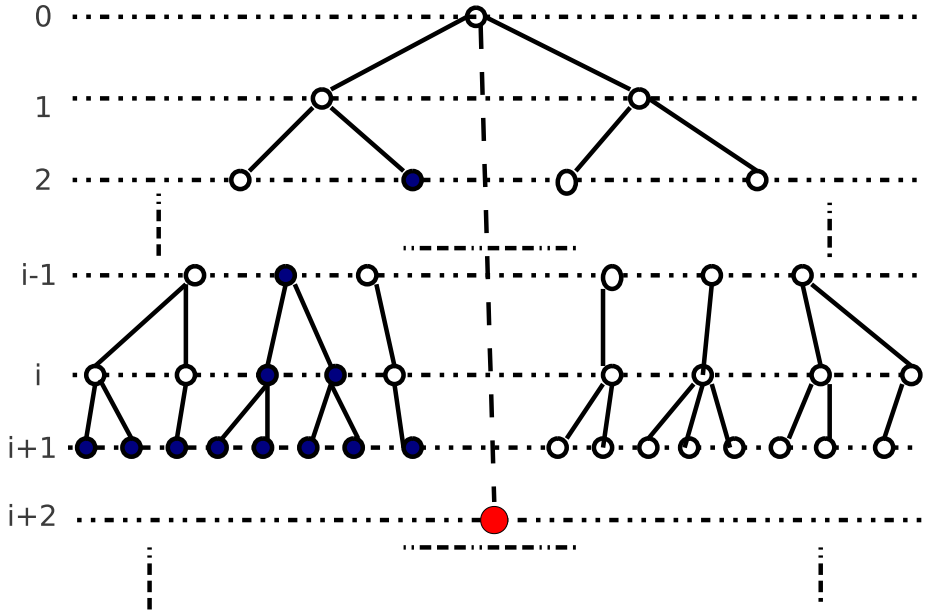


Figure 1.8: Incremental approach with One Desired Node.

Therefore, one promising problem in this thesis is to find the minimum depth k_{min}, which will contain solutions in the search tree.

Since each step has to be instantiated to reach the desired marking, the timed firing sequence will be given by this resolved path. *Therefore timed firing sequences in TPNs and the reachability problem are solved together by our incremental approach.*

In fact, the sequence of timed steps $\psi_1, \ldots, \psi_{i+2}$ will represent the whole behavior of the TPNs with $i + 2$ steps. Each step can be represented as *variables* and relations between steps can be represented as constraints.

Then exploring the underlying reachability graph of TPNs is actually replaced at resolving constraints. To solve constraints that represent relations between different timed steps, we propose to use constraint programming to be the constraint solver.

1.3.4 An Approach based on Constraint Programming

In this thesis, we propose to develop a CP model to analyze TPNs. In fact, the problem of generating firing sequences in TPNs can be seen as a combinatorial satisfactory problem (CSP). The CP can solve these CSPs with powerful analyzing tools and is used to the context of verification techniques.

CP is a powerful paradigm for solving combinatorial search problems that draws on a wide range of techniques from artificial intelligence, computer science, and operations research. The user declaratively states the constraints on the feasible solutions for a set of decision variables [RBW06].

In CP, the *incremental model* can be easily translated into CP model, in which conditional constraints are very powerful in modeling non-linear equations.

Take figure (1.8) as an example, all states are represented by the sequences of timed steps $\psi_1, \ldots, \psi_{i+2}$ implicityly. In CP, these timed steps can be seen as variables and relations between these steps are seen as constraints.

When searching for a solution, CP can control the search process based on search strategies. Search strategies can lead solver to the desired marking more quickly, which can also reduce the influence of combinatorial explosion.

In figure (1.8), in the best situation, if the $k_{min} = i{+}2$ is first given and a better search strategy is applied, solver can find the solution without meeting deadends. This means that only the path leading to the desired marking is explored, as shown in figure (1.8).

In this thesis, many techniques are applied based on CP techniques to improve the efficiency at searching for solutions of reachabillity problems.

1.4 Summary

1.4.1 Contribution

The main contribution in this thesis are as follows:

- An incremental model is developed based on the idea of BMC and logical abstraction technique to represent the whole behavior of TPNs. It allows to an implicit traversal the behavior of TPNs without building the whole

construction. The correctness and completeness of this model are proved based on the model of [CC88].

- Based on this incremental model, the problem of generating firing sequences in TPNs can be solved. Then several incremental search algorithms are introduced to improve the search efficiency.

- The incremental model is successfully translated into CP model. During modeling, non-linear equations are naturally represented by conditional constraint, and several improvements are developed as constraints. It means that our model has more extendibility for expressing properties and specific needs in real systems.

- With the help of CP, several effective search strategies are developed to search for one solution or all solutions. Search strategies are developed based on the structure information of TPNs, which can search for solutions more efficiently.

- We apply our incremental model to the reconfiguration of manufacturing systems. Especially, the token identification technique for bounded TPNs is developed. This technique can help to distinguish tokens in TPNs more generally, for example, without safe behavior limitation.

We can conclude that a formal method for generating firing sequences in TPNs is developed, which involves the establishment of objectives, the modeling of real-time complex systems, the solving of reachability problems in TPNs. Especially, many specific problems in practical systems (like token confusion issue in manufacturing system) can be modeled and solved based on our incremental model.

1.4.2 Organization

Chapter (1) is a literature survey, which provides an overview of problems and challenges, methods for reducing combinatorial explosion, and respect researches in the field of TPNs. More precisely, several methods are introduced to reduce combinatorial explosion.

Chapter (2) develops an incremental model for TPNs, which is the basic model in this thesis. Firstly, the formal definition of TPNs and their terminology are introduced based on [CC88]. *Controlled Execution* introduced in [CC88] are defined for presenting the TPN instantaneous state and reachability problem. Secondly, to reduce the influence of *combinatorial explosion* in TPN, the incremental approach is introduced based on an *implicit traversal* of reachability graph of TPN, which does not need its construction. This is done by considering a unique sequence of timed steps growing incrementally to represent exactly the total behavior of TPN. Finally, several incremental search algorithms are introduced to improve the efficiency at the search process.

Chapter (3) translates the incremental model into a CP model, develops several search strategies to solve reachability problems more efficiently and apply the linearization technique for improving efficiency. The terminology of CP is briefly introduced. For analyzing systems with CP, *Modeling* and *Solving* are two main parts. In the modeling part, CP can develop constraints to express relations and specific needs for systems, for example token confusion issues in chapter (4). Especially, non-linear relations can be expressed by conditional constraints. In the solving part, we develop several search strategies to improve the efficiency for searching. Finally, a linearization technique is applied to linearize non-linear equations at the solving part, since variables in conditional constraints must be labeled at last.

Chapter (4) applies the incremental approach to a realistic issue concerning manufacturing systems: the problem of reconfiguration of manufacturing systems. Firstly, main issues of reconfigurable transport system are given based on decomposing it into different TPN model, such as *operating sequence*, *transport system*, etc. Then token confusion issues are given, and token identification technique for ordinary PN is introduced. Secondly, a token identification technique for TPNs with safe behavior is developed. Thirdly, to remove the limitation of safe behavior, a token identification technique for bounded TPNs is developed. In fact, each kind of tokens form a TPN layer. Different kinds of tokens are connected through firing sequences. Fourthly, a firing priority vector is defined to avoid loops. Finally, benchmarks are given for presenting efforts of our approaches.

Chapter (5) concludes the work of this thesis and gives perspectives. Main approaches are concluded by expressing the main ideas of this thesis. The application of reconfiguration transport system is addressed by our approaches. Several perspectives are given to show possible work to improve our model.

Chapter 2

Incremental Approach for Timed Petri Nets

The objective of this chapter is to formulate an ***incremental model*** of the behavior of TPNs avoiding to build their whole reachability graph, which can deal with the *combinatorial explosion* problem.

Our approach is based on an *implicit traversal* of the underlying reachability graph of TPNs instead of building the whole construction. This is done by considering a unique sequence of timed steps growing incrementally to represent exactly the behavior of the TPN.

This incremental model allow us developing efficient techniques for analyzing TPN. For instance, an ***incremental search*** is developed to solve reachability problems in TPNs.

In the first section, we study the behavior of TPNs in terms of reachability relations between states. We formally define TPN and TPN states following the work of [CC88], where the ***residual duration vector*** is introduced for expressing *missing* tokens during transition firing. Then the TPN state equation and TPN reachability problem are given following the controlled executions defined in [CC88]. As it is not possible to explore the reachability graph exhaustively due to the well known problem of combinatorial explosion, we propose to develop an incremental model to solve the reachability problem in TPNs.

The second section is dedicated to the formal definition of an incremental model of the behavior of TPNs. The concepts of timed step and timed step **firings**

are defined to form the foundation ideas of the incremental model. Following these concepts, the instantaneous state is given combined with fireable conditions and firing sets definition. By proving the equivalence between timed step firings and *controlled executions*, we show how this model incrementally capture the behavior of the associated PN. Therefore we can conclude that every state that is reachable from initial state can be reached by a sequence of timed step firings.

Finally, we introduce several incremental search algorithms to generate firing sequences for TPNs. The depth of the search will influence the efficiency of solving problems. Therefore, two kind of sub-problems are defined: *fixed depth reachability problem* and *shortest length reachability problem*. Then naive algorithms and jump search algorithms are introduced to present the mechanism of incremental search and improve its efficiency.

2.1 Timed Petri Nets Reachability Problem

TPN has been first introduced in [Ram74]. It mainly adds a fixed length time duration to each transition, which is more practical in time critical systems. The following presentation is adapted from [Chr84]. We start by giving an informal introduction on TPNs.

2.1.1 Informal Presentation

TPNs correspond to Places/Transitions PNs where a *duration* $d(t) \in \mathbb{N}^+$ is associated to each transition t. A TPN has the same representation as a PN, to which is added a *labelling* on transitions. An example of TPN is given in figure (2.1). We have: $d(t_1) = 3$, $d(t_2) = 4$, $d(t_3) = 5$, $d(t_4) = 2$, $d(t_5) = 1$, $d(t_6) = 1$.

The firing durations associated to transitions modify the *marking validity conditions*. As soon as durations are associated to transitions, the PN acts as if tokens *"disappeared"* at the time the transition is fired, and then *"reappeared"* after a delay corresponding to the duration of the fired transition. Thus, the marking of a TPN evolves with the occurrences of an external timer. For instance, let us consider the firing sequence of TPNs showed in table. (2.1). At date 1, the transition t_1 (duration: 3 *t.u.*) is fired. Then the transition t_4 (duration: 2 *t.u.*) is fired at

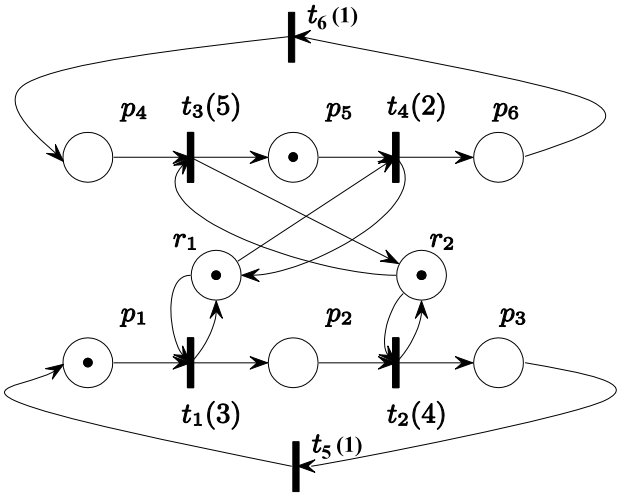


Figure 2.1: A Timed Petri Net

date 5, etc. The firing sequence is given as: $(t_1, 1), (t_4, 5), (t_2, 6), (t_5, 11), (t_1, 13)$. The evolution of marking with time is also given in table (2.1). Note that one could have fired transition t_4 at date 4, since the resource r_1 had been released at the end of the firing of transition t_1. However, the same transition was not fireable at date 3, since the firing of t_1 was not finished.

The firing and ending dates of transitions play a fundamental role in the behavior of the TPN. It is thus necessary to *adapt* the firing equations according to these firing dates. In order to respect the underlying semantics of PN, a *firing sequence* is said to be *feasible* if and only if, at any time, the transient marking reached is made of *non negative* components.

In [CC88], authors propose *Controlled Executions* to govern the behavior of TPNs. Firing one transition is first considered with an increasing sequence of firing dates. Take table (2.1) as an example, the transition t_1 is fired at dates 1 and 13, [CC88] denote these firing dates as $u_1^{t_1} = 1$ and $u_2^{t_1} = 13$. A family of firing sequences of all transitions is called *controlled executions* (CE).

2.1.2 Timed Petri Nets Terminology

Definition 2.1 (TPN – Timed Petri Net). *A Timed Petri Net [Ram74]* $(R = (\mathbb{P}, \mathbb{T}, C^+, C^-, m_0), d)$*, with its initial marking* m_0 *is a bipartite directed graph*

	Date	Marking $(p_1, p_2, \ldots, p_6, r_1, r_2)$	Controlled Execution
Initial date	0	$(1,0,0,0,1,0,1,1)$	
Firing of $t_1 \rightarrow$	1	$(0,0,0,0,1,0,0,1)$	$u_1^{t_1} = 1$
	2	$(0,0,0,0,1,0,0,1)$	
	3	$(0,0,0,0,1,0,0,1)$	
End of $t_1 \rightarrow$	4	$(0,1,0,0,1,0,1,1)$	
Firing of $t_4 \rightarrow$	5	$(0,1,0,0,0,0,0,1)$	$u_1^{t_4} = 5$
Firing of $t_2 \rightarrow$	6	$(0,0,0,0,0,0,0,0)$	$u_1^{t_2} = 6$
End of $t_4 \rightarrow$	7	$(0,0,0,0,0,1,1,0)$	
	8	$(0,0,0,0,0,1,1,0)$	
	9	$(0,0,0,0,0,1,1,0)$	
End of $t_2 \rightarrow$	10	$(0,0,1,0,0,1,1,1)$	
Firing of $t_5 \rightarrow$	11	$(0,0,0,0,0,1,1,1)$	$u_1^{t_5} = 11$
End of $t_5 \rightarrow$	12	$(1,0,0,0,0,1,1,1)$	
Firing of $t_1 \rightarrow$	13	$(0,0,0,0,0,1,0,1)$	$u_2^{t_1} = 13$

Table 2.1: Firing Semantics and Controlled Execution of figure (2.1)

where:

- $\mathbb{P}$ *and* $\mathbb{T}$ *are two finite sets of nodes denoted respectively* places *and* transitions *with* $|\mathbb{P}| = \mathrm{M}$ *and* $|\mathbb{T}| = \mathrm{N}$. *Places are represented as circles and transitions as rectangles. Places generally represent conditions, and transitions represent events, see [Mur89].*

- *Incidence matrices* C^-, C^+ *and* $C \in \mathbb{N}^{\mathbb{M}\times\mathbb{N}}$ *(with* $C = C^+ - C^-$*) define the weighted flow function which associates to each arc* (p_i, t_j) *or* (t_j, p_i) *its weight* C_{ij}^- *or* C_{ij}^+. *When there is no arc between place* p_i *and transition* t_j, *then we have:* $C_{ij}^- = C_{ij}^+ = 0$; *The notations upstream and downstream denote respectively the predecessors and successors of a node of the net*

- $m_0 : \mathbb{P} \rightarrow \mathbb{N}$ *associates to each place* $p \in \mathbb{P}$ *an integer* $m_0(p)$ *called the* initial marking *of the place* p. *Markings are represented as full dots called* tokens *inside places;*

- $d : \mathbb{T} \rightarrow \mathbb{N}^*$ *is a mapping associating a duration to each transition of the net.*

In the following, we use linear algebra formulations to make the presentation

more concise. For instance, $\overrightarrow{e_{t_k}}$ denotes the Parikh vector associated to the transition t_k, the k$^{\text{th}}$ component of which takes the value 1 and others 0. $\overrightarrow{0_d}$ denotes the empty vector of dimension d. $\overrightarrow{I_d}$ denotes the unit vector of dimension d.

Note that we will use notations $d(t_j)$ or d_j to represent the j^{th} element in the duration vector.

One could more generally consider *rational valued* durations. Nevertheless, after having them reduced to the same denominator, and by reasoning over numerators, it is the same as if durations were *integer valued*.

The transition firing semantics in TPN forbids ***reentrance***. In other words, it is not possible to fire again a transition that has not yet *finished* to be fired (*mono-server semantics*).

Timed Petri Net State

In ordinary PN (PN without timing labels), a transition t_j is said to be *fireable* from a given marking m if and only if its *upstream* places have enough tokens: $\overrightarrow{m} \geqslant C^- \cdot \overrightarrow{e_{t_j}}$. When the transition is fired, a new marking m' is produced, such that: $\overrightarrow{m'} = \overrightarrow{m} + C \cdot \overrightarrow{e_{t_j}}$. The state of this net is only represented by its marking.

As presented in section (2.1.1), the firing and ending dates of transitions play a fundamental role in the behavior of the TPN. It is thus necessary to *adapt* the firing equations according to these firing dates. One can associate a unique *residual duration* to each transition without any possible confusion between several concurrent transition activations. The residual duration vector is associated to the marking of a TPN to define its full state.

Definition 2.2 (TPN State). *Let (R, d) be a TPN. Its* state $s = (\overrightarrow{s_m}, \overrightarrow{s_r})$ *is given by:*

- *its classical* marking vector $\overrightarrow{s_m} \in \mathbb{N}^{\text{M}}$*, associating to each place its number of tokens;*
- *a* residual durations vector $\overrightarrow{s_r} \in \mathbb{N}^{\text{N}}$*, associating to each* active *transition its remaining duration, and zero if the transition is not active.*

The set of all states of a TPN is denoted by $\mathscr{S}(R, d)$ and the residual duration vector is defined at date i as $\overrightarrow{s_{r_i}} = (s_{r_i}(t_1), s_{r_i}(t_2), \ldots, s_{r_i}(t_{\text{N}}))^{\intercal} \in \mathbb{N}^{\text{N}}$.

2.1.3 Timed Petri Nets State Equation

The fundamental concept that governs TPN behavior is the *controlled execution*, introduced by [CC88], which associates to each transition the sequence of its successive firing dates.

Definition 2.3 (CE – Controlled Execution). *Let (R, d) be a TPN and $t \in \mathbb{T}$ a transition. A firing sequence for the timed transition t: $(u_k^t) = u_1^t, \ldots, u_{\mathrm{K}_t}^t \in \mathbb{N}$ is an increasing sequence of firing dates, such that:*

$$\forall k \in [\![1, \mathrm{K}_t - 1]\!], u_k^t + d(t) \leqslant u_{k+1}^t \tag{2.1}$$

A controlled execution is a family $(u_k^t)_{t\in\mathbb{T},k\in[\![1,K_t]\!]}$ of firing sequences for all transitions in TPN.

Note that in the previous definition, equation (2.1) is used to forbid *reentrance*. For any transition t, K_t and $u_{\mathrm{K}_t}^t$ may be very big, we denote $v_{\max}$ as the ending date of the last firing in the CE: $v_{\max} = \max_{t\in\mathbb{T}} \left(u_{\mathrm{K}_t}^t + d(t)\right)$. After $v_{\max}$, the state of the TPN under the considered CE will never change.

The formal expression of a CE is used to define several *characteristic vectors* allowing to verify the feasibility of a CE. We assume that no transition is active at the initial state to simplify the formulation.

Definition 2.4 (Characteristic Vectors of Controlled Executions). *Let (R, d) be a TPN with its initial state $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ given at initial date 0 and $(u_k^t)_{t\in\mathbb{T},k\in[\![1,K_t]\!]}$ a controlled execution. Let $v \in [\![0, v_{\max}]\!]$. Two characteristic vectors are associated to (u_k^t) in the following way:*

- $\overrightarrow{N(v)} \in \mathbb{N}^{\mathrm{N}}$ *is the vector corresponding to the number of firings that* started *within the interval $[0, v]$, defined by $\overrightarrow{N(v)}\Big|_t = card\left(\{u_k^t, k \in [\![1, \mathrm{K}_t]\!] \mid u_k^t \leqslant v\}\right)$;*
- $\overrightarrow{F(v)} \in \mathbb{N}^{\mathrm{N}}$ *is the vector corresponding to the number of firings that* ended *within the interval $[0, v]$, defined by $\overrightarrow{F(v)}\Big|_t = card\left(\{u_k^t, k \in [\![1, \mathrm{K}_t]\!] \mid u_k^t + d(t) \leqslant v\}\right)$.*

To illustrate how CE works with these characteristic vectors. Table. (2.2) is given corresponding to figure (2.1). $\overrightarrow{N(v)}$ represents all transitions that fired from

	V	$\overrightarrow{s_m}$	CE	$\overrightarrow{N(v)}$	$\overrightarrow{F(v)}$
Initial date	0	$(1,0,0,0,1,0,1,1)$		$\emptyset$	$\emptyset$
Firing of $t_1 \rightarrow$	1	$(0,0,0,0,1,0,0,1)$	$u_1^{t_1} = 1$	$(1,0,0,0,0,0)$	$(0,0,0,0,0,0)$
	2	$(0,0,0,0,1,0,0,1)$		$(1,0,0,0,0,0)$	$(0,0,0,0,0,0)$
	3	$(0,0,0,0,1,0,0,1)$		$(1,0,0,0,0,0)$	$(0,0,0,0,0,0)$
End of $t_1 \rightarrow$	4	$(0,1,0,0,1,0,1,1)$		$(1,0,0,0,0,0)$	$(1,0,0,0,0,0)$
Firing of $t_4 \rightarrow$	5	$(0,1,0,0,0,0,0,1)$	$u_1^{t_4} = 5$	$(1,0,0,1,0,0)$	$(1,0,0,0,0,0)$
Firing of $t_2 \rightarrow$	6	$(0,0,0,0,0,0,0,0)$	$u_1^{t_2} = 6$	$(1,1,0,1,0,0)$	$(1,0,0,0,0,0)$
End of $t_4 \rightarrow$	7	$(0,0,0,0,0,1,1,0)$		$(1,1,0,1,0,0)$	$(1,0,0,1,0,0)$
	8	$(0,0,0,0,0,1,1,0)$		$(1,1,0,1,0,0)$	$(1,0,0,1,0,0)$
	9	$(0,0,0,0,0,1,1,0)$		$(1,1,0,1,0,0)$	$(1,0,0,1,0,0)$
End of $t_2 \rightarrow$	10	$(0,0,1,0,0,1,1,1)$		$(1,1,0,1,0,0)$	$(1,1,0,1,0,0)$
Firing of $t_5 \rightarrow$	11	$(0,0,0,0,0,1,1,1)$	$u_1^{t_5} = 11$	$(1,1,0,1,1,0)$	$(1,1,0,1,0,0)$
End of $t_5 \rightarrow$	12	$(1,0,0,0,0,1,1,1)$		$(1,1,0,1,1,0)$	$(1,1,0,1,1,0)$
Firing of $t_1 \rightarrow$	13	$(0,0,0,0,0,1,0,1)$	$u_2^{t_1} = 13$	$(2,1,0,1,1,0)$	$(1,1,0,1,1,0)$

Table 2.2: Controlled execution and its characteristic vectors

date 0 until date v. $\overrightarrow{F(v)}$ represents all transitions that finished firing from date 0 to date v.

The state of a TPN is modified under a CE in the following way.

Definition 2.5 (Instantaneous State of a TPN under a Controlled Execution). *Let (R, d) be a TPN with its initial state $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ given at date 0 and $(u_k^t)_{t\in\mathbb{T},k\in[\![1,K_t]\!]}$ a controlled execution. Let $v \in [\![0, v_{\max}]\!]$. The instantaneous state $s_v = \left(\overrightarrow{s_m(v)}, \overrightarrow{s_r(v)}\right)$ at date v is given by:*

$$\overrightarrow{s_m(v)} = \overrightarrow{s_{m_0}} + C^+ \cdot \overrightarrow{F(v)} - C^- \cdot \overrightarrow{N(v)} \tag{2.2}$$

$$\forall t \in \mathbb{T}, \left.\overrightarrow{s_r(v)}\right|_t = \begin{cases} u_k^t + d(t) - v & \text{if } \exists k \in [\![1, \mathrm{K}_t]\!] \text{ s.t. } v \in [\![u_k^t, u_k^t + d(t)[\![\\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Informally, in the previous definition, the quantity $C^+ \cdot \overrightarrow{F(v)}$ corresponds to the tokens produced by the firings of transitions that ended before *the date v*. Those tokens can be used to fire transitions *at date v*. The quantity $C^- \cdot \overrightarrow{N(v)}$ corresponds to the tokens used by the firings of transitions that started *until the date v*. Thus, the quantity $\overrightarrow{s_m(v)}$ corresponds exactly to the tokens remaining in the TPN at date v.

Obviously, like for Place/Transitions PNs, even if each transition is independently fireable at every date, the full CE is not necessarily valid as a whole since a token may be used by several transitions at the same time. Thus, a condition for a CE to be feasible is given below.

Definition 2.6 (Feasible Controlled Execution). *Let (R, d) be a TPN with its initial state $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ given at date 0 and $(u_k^t)_{t\in\mathbb{T},k\in[\![1,K_t]\!]}$ a controlled execution. This controlled execution is said to be feasible iff:*

$$\forall v \in [\![0, v_{\max}]\!], \overrightarrow{s_m(v)} \geqslant \overrightarrow{0_{\mathrm{M}}} \tag{2.4}$$

The above condition means that there must be enough tokens so that transitions may fire simultaneously.

2.1.4 Timed Petri Nets Reachability Problem

Using the previous notations, the TPNs reachability problem consists in searching for a feasible CE allowing to reach a given final state from the initial state.

Definition 2.7 (TPN Reachability Problem). *Let (R, d) be a TPN with its initial state $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ given at date 0. Let $s_f = \left(\overrightarrow{s_{m_f}}, \overrightarrow{0_{\mathrm{N}}}\right)$ be a target state. The reachability problem of TPNs consists in finding a CE $(u_k^t)_{t\in\mathbb{T},k\in[\![1,K_t]\!]}$ such that $s_{v_{\max}} = \left(\overrightarrow{s_m(v_{\max})}, \overrightarrow{s_r(v_{\max})}\right) = s_f$.*

Indeed, let us consider for instance the TPN of figure (2.1). One remarks obviously that solving a reachability problem between markings $\overrightarrow{s_{m_0}} = \{1, 0, 0, 1, 0, 0, 1, 1\}$ and $\overrightarrow{s_{m_f}} = \{0, 0, 1, 0, 0, 1, 1, 1\}$ (where $\overrightarrow{s_{r_0}} = \overrightarrow{s_{r_f}} = \overrightarrow{0_N}$) means exactly finding a firing sequence leading from the initial state to the final state

Several approaches have been proposed to solve the TPN reachability problem, either by restricting their study to a subclass of TPN, like Timed Event Graphs, either by using dedicated heuristics. A complete bibliography can be found in [Ric00].

In this thesis, we propose to use an incremental approach to express the behavior of TPNs, avoiding to build the whole reachability graph, which is more suitable for dealing with combinatorial explosion.

2.2 Incremental Approach

2.2.1 Informal Presentation

In this section, we propose to build an *incremental* model corresponding to an increasing number of *firing dates*. We introduce the way for updating states of TPNs with incremental approach by introducing the logical relationship between *Marking vector* $\overrightarrow{s_{m_i}}$ and *Residual duration vector* $\overrightarrow{s_{r_i}}$.

Firstly, the residual duration vector is just defined for recording *disappeared* tokens during transition firing, and the *steady* state is determined by the marking vector. For marking vector, we do not need to update every time. Let us consider table (2.1) as example, the marking is the same at dates 2 and 3 as the date 1, and the residual duration of fired transition will reduce regularly. Based on the analysis in this paragraph, **the states can be updated when the marking vector is changed**, like dates 4 and 7 where tokens are *added* to places, and date 1 and 5 where tokens are *removed* from places. So a new symbol Δ_i is given to represent the time elapsed between two firings.

Second, *removing* tokens can be only done when transitions are fired by satisfying the firing policy which means all *upstream* places must have enough tokens to be removed. But *adding* tokens to the *downstream* places do not have such limit, thus intermediate markings remain positive between firing dates if they were positive at each firing date. Based on the analysis above, **the states will be only updated at the moment of new firings** (*removing* tokens from places). A new symbol v_k is introduced as the date of the k_{th} firing time. Then the TPN state can be written as $s_k = (\overrightarrow{s_{m_k}}, \overrightarrow{s_{r_k}})$.

So, contrary to Controlled Executions, we do not need to consider intermediate states between two firings to verify the validity of a step sequence. Such consideration allows to reduce the number of states to be considered, thus reducing the combinatorial explosion.

The key idea of incremental model is to consider the evolution of a TPN ***step by step***. The expression *step* is borrowed from [JK91].

A synoptic of notations is given in table (2.3). We express at any firing date v_{k+1} (denoted by the time Δ_{v_k} elapsed from the previous firing) the induced

modifications on the vectors $\overrightarrow{s_m}(v_{k+1})$ and $\overrightarrow{s_r}(v_{k+1})$ by the firing of a step $\sigma(v_{k+1})$. To be more concise, we denote intermediate expressions using the index of the corresponding firing date : $\overrightarrow{s_m}(v_k) \equiv \overrightarrow{s_{m_k}}$, $\sigma(v_k) \equiv \sigma_k$, etc.

s_{m_0}	$\longrightarrow$	$s_{m_1} \dots s_{m_k}$	$\longrightarrow$	$s_{m_{k+1}} \dots s_{m_{\mathsf{K}-2}}$	$\longrightarrow$	$s_{m_{\mathsf{K}-1}}$	$\longrightarrow$	$s_{m_{\mathsf{K}}} = s_{m_f}$
	σ_1		σ_{k+1}		$\sigma_{\mathsf{K}-1}$		$\sigma_{\mathsf{K}} = 0_N$	
$v_0 = 0$	$v_1 =$		$v_{k+1} =$		$v_{\mathsf{K}-1} =$		$v_{max} =$	
	$v_0 + \Delta_{v_0}$		$v_k + \Delta_{v_k}$		$v_{\mathsf{K}-2} + \Delta_{\mathsf{K}-2}$		$v_{\mathsf{K}-1} + \Delta_{\mathsf{K}-1}$	
s_{r_0}	$\longrightarrow$	$s_{r_1} \dots s_{r_k}$	$\longrightarrow$	$s_{r_{k+1}} \dots s_{r_{\mathsf{K}-2}}$	$\longrightarrow$	$s_{r_{\mathsf{K}-1}}$	$\longrightarrow$	$s_{r_{\mathsf{K}}} = s_{r_f}$

Table 2.3: Synoptic Table of Notations.

In table (2.3), processes of *removing* tokens from upstream places and *adding* tokens to downstream places, which are done at the same time of transition firings in PNs, are splitted into two different steps. For example, if one transition t is fired at date v_1, tokens in upstream places of t will be removed at the same date v_1. Since tokens will be hold in the firing transition for a fixed length of time and we only update states when new firings happen, tokens will be added to downstream places at next firing step.

Note that we restrict ourselves to TPNs *without immediate transitions* (i.e. $\forall t \in \mathbb{T}, d(t) > 0$), which is not so restrictive in real world practice.

2.2.2 Timed Steps Firings

Definition 2.8 (Step). *Let R be a PN. A step [JK91] is a multiset over the set of transitions $\mathbb{T}$. We denote by $\mathbb{T}^*$ the set of steps built over $\mathbb{T}$.*

Informally, a step is a set that can contain several copies of the same element, e.g. $\{t_1, t_1, t_2\}$, which we would note hereafter simply $2 \cdot t_1 + t_2$. We associate a step $\sigma = \sum_{j=1}^{\mathrm{N}} \alpha_j \cdot t_j$ and its Parikh vector $\overrightarrow{\sigma}$ in the classical way, as a linear combination with non negative integer coefficients α_j of the Parikh vectors of each transition, i.e. $\overrightarrow{\sigma} = \sum_{j=1}^{\mathrm{N}} \alpha_j \cdot \overrightarrow{e_{t_j}}$. A step is said *empty*, when $\sigma = \varnothing$, i.e. when $\forall j \in [\![1, \mathrm{N}]\!], \alpha_j = 0$.

Note that a PN step can contain the same transition more than once, corresponding to *transition reentrance*. Thus, when working with TPNs, steps would only mean that several *different* transitions are considered to be fired at the same time.

Definition 2.9 (Timed Step). *Let (R, d) be a TPN. A* timed step *is a pair $\psi = (\overrightarrow{\sigma}, v)$ such that:*

- *v is a date $\in \mathbb{N}$;*
- *$\sigma \in \{0,1\}^{\mathbb{T}}$ is a function denoting a set of concurrent transition firings at date v, using the monoserver semantics. We denote by $\overrightarrow{\sigma_k}$ the Parikh vector of* new fired transitions *at date v_k.*

To express the behavior of a TPN between two dates v_k and v_{k+1} into a state equation, we define sets of *finishing* and *still firing* transitions at date v_k.

Definition 2.10 (FS – Firing Sets). *Let (R, d) be a TPN with its initial state $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathbb{N}}}\right)$ given at date v_0. Let $\psi_i = (\overrightarrow{\sigma_i}, v_i)_{i \in [\![1,k]\!]}$ be a sequence of k* timed step *firings, and $v_{k+1} \in \mathbb{N}$ s.t. $v_{k+1} > v_k$. Two firing sets T^f_{k+1} and T^s_{k+1} are associated to the date v_{k+1} in the following way:*

- *The set of* finishing *transitions $T^f_{k+1} \in \mathbb{T}^{\mathsf{N}}$ denotes transitions active at date v_k that are no more firing at date v_{k+1}. Formally:*

$$T^f_{k+1} = \Bigg\{ \quad t \in \mathbb{T}, \exists i \in [\![1,k]\!] \text{ s.t. } \sigma_i(t) = 1 \\ \wedge \quad v_k < \max_{i \in [\![1,k]\!]} \{v_i : \sigma_i(t) = 1\} + d(t) \leqslant v_{k+1} \Bigg\} \tag{2.5}$$

- *The set of* still firing *transitions $T^s_{k+1} \in \mathbb{T}^{\mathsf{N}}$ denotes transitions active at date v_k that are still firing at date v_{k+1}. Formally:*

$$T^s_{k+1} = \Bigg\{ \quad t \in \mathbb{T}, \exists i \in [\![1,k]\!] \text{ s.t. } \sigma_i(t) = 1 \\ \wedge \quad v_k < v_{k+1} < \max_{i \in [\![1,k]\!]} \{v_i : \sigma_i(t) = 1\} + d(t) \Bigg\} \tag{2.6}$$

In these equations, $\max\limits_{i \in [\![1,k]\!]} \{v_i : \sigma_i(t) = 1\}$ denotes the latest firing date of t. Using such notations, we express the state equation for TPN using timed step firings in the following way: for a step to be fireable, its preceding marking must contain enough tokens so that each transition of the step may consume their *own* tokens, as described in the following definition.

Definition 2.11 (Timed Step Firings). *Let (R, d) be a TPN with its initial state $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathbb{N}}}\right)$ given at date v_0. Let $\psi_i = (\overrightarrow{\sigma_i}, v_i)_{i \in [\![1,k]\!]}$ be a sequence of k*

timed step firings leading to state $s_k = (\overrightarrow{s_{m_k}}, \overrightarrow{s_{r_k}})$ *at date* v_k. *Let* $v_{k+1} \in \mathbb{N}$ *s.t.* $\Delta_{v_k} = v_{k+1} - v_k > 0$. *The timed step* $\psi_{k+1} = (\overrightarrow{\sigma_{k+1}}, v_{k+1})$ *is fireable from* s_k *iff:*

$$\forall t \in \mathbb{T} \text{ s.t. } \sigma_{k+1}(t) = 1, \quad s_{r_k}(t) \leqslant \Delta_{v_k} \tag{2.7}$$

$$C^- \cdot \overrightarrow{\sigma_{k+1}} \leqslant \overrightarrow{s_{m_k}} + C^+ \cdot \sum_{t_j \in T^f_{k+1}} \overrightarrow{e_{t_j}} \tag{2.8}$$

If this condition is satisfied, the new state $s_{k+1} = (\overrightarrow{s_{m_{k+1}}}, \overrightarrow{s_{r_{k+1}}})$ *reached at date* v_{k+1} *from* s_k *by the firing of* $\psi_{k+1} = (\overrightarrow{\sigma_{k+1}}, v_{k+1})$ *is defined as:*

$$\overrightarrow{s_{m_{k+1}}} = \overrightarrow{s_{m_k}} - C^- \cdot \overrightarrow{\sigma_{k+1}} + \sum_{t_j \in T^f_{k+1}} C^+ \cdot \overrightarrow{e_{t_j}} \tag{2.9}$$

$$\overrightarrow{s_{r_{k+1}}} = \sum_{t_j \in T^s_{k+1}} \left(s_{r_k}(t_j) - \Delta_{v_k}\right) \cdot \overrightarrow{e_{t_j}} + \sum_{t_j \in \mathbb{T}} d(t_j) \cdot \sigma_{k+1}(t_j) \cdot \overrightarrow{e_{t_j}} \tag{2.10}$$

$s_k[\psi_{k+1}\rangle s_{k+1}$ *denote that* ψ_{k+1} *is firable from* s_k, *and leads to* s_{k+1}.

The above definition follows the firing semantics of TPNs described above, from the point of view of a punctual firing between two markings. Expression $C^- \cdot \overrightarrow{\sigma_{k+1}}$ represents the number of tokens removed from places upstream to the new fired transitions σ_{k+1} at the date v_{k+1}. Expression $\sum_{t_j \in T^f_{k+1}} C^+ \cdot \overrightarrow{e_{t_j}}$ represents tokens added to the places downstream transitions finished at date v_{k+1}. Expression $\sum_{t_j \in T^s_{k+1}} \left(s_{r_k}(t_j) - \Delta_{v_k}\right) \cdot \overrightarrow{e_{t_j}}$ represents the update of residual durations for transitions that were active at date v_k and are still active at date v_{k+1}. The time elapsed corresponds to $\Delta_{v_k} = v_{k+1} - v_k$ units of time. Expression $\sum_{t_j \in \mathbb{T}} d(t_j) \cdot \sigma_{k+1}(t_j) \cdot \overrightarrow{e_{t_j}}$ represents the *new residual durations* coming from the execution of the new fired transitions in σ_{k+1} at the date v_{k+1}. Finally, equation (2.7) means that a transition fired within a step must not be *still firing*, in order to comply with the monoserver semantics.

At last, the update of the residual durations vector at equation (2.10) is made as follows:

- If the transition t is fired in the step ψ_{k+1}, its duration is used to initialize the corresponding component of the residual vector;

- If a transition that was active at date v_k and is *still firing* at date v_{k+1}, its residual duration is updated by taking into account the time elapsed from the previous date;
- Otherwise, if a transition was not active at date v_k or finished before date v_{k+1} and it is not fired in the step ψ, its residual duration will be 0.

As above, we will use the notations $s[\psi\rangle$, $s_0[\psi_1\rangle s_1$, $s_0[\psi_1\psi_2\dots\psi_k\rangle$ and $s_0[\psi_1\psi_2\dots\psi_k\rangle s_k$ to indicate that a timed step firing sequence is fireable, and the state obtained in each case.

2.2.3 Towards an Incremental Model

We give below the main propositions concerning the use of timed steps in the context of TPN.

Proposition 2.12 (Equivalence between CE and timed step firing sequences). *Let (R,d) be a TPN with its initial state $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ given at date $v_0 = 0$. Let $s_f = \left(\overrightarrow{s_{m_f}}, \overrightarrow{0_{\mathrm{N}}}\right) \in \mathscr{S}(R,d)$ be a state.*

There exists a feasible controlled execution allowing to reach s_f at date $v_{\max}$ from $s_0 \Leftrightarrow$ there exists a sequence of timed step firings allowing to reach s_f at date v_{max} from s_0. i.e.

$$\begin{array}{l} \exists \mathrm{K} \in \mathbb{N}, \\ \exists v_1, v_2, \dots, v_{\mathrm{K}} \in \mathbb{N} \\ \exists s_1, s_2, \dots, s_{\mathrm{K}} \in \mathscr{S}(R,d). \\ \exists \psi_1 = (\overrightarrow{\sigma_1}, v_1), \psi_2, \dots, \psi_{\mathrm{K}} = (\overrightarrow{\sigma_{\mathrm{K}}}, v_{\mathrm{K}}), \\ \qquad\qquad \psi_{\max} = (\overrightarrow{0_{\mathrm{N}}}, v_{\max}) \in \mathbb{T}^*_{TPN} \end{array} \quad s.t. : \left\{ \begin{array}{r} \forall k \in [\![1, \mathrm{K}]\!], s_{k-1}[\psi_k\rangle s_k \\ s_{\mathrm{K}}[\psi_{\max}\rangle s_f \end{array} \right. \tag{2.11}$$

The full proof takes 8 pages. Thus we put it in appendix A. Main ideas in the proof are as follow:

- $(\Rightarrow)$: We assume that $(u_k^t)_{t\in\mathbb{T}, k\in[\![1,K_t]\!]}$ be a CE such that $s_{v_{\max}} = \left(\overrightarrow{s_m(v_{\max})}, \overrightarrow{s_r(v_{\max})}\right) = s_f$. Then we buid a step sequence from (u_k^t) as follows. Let $v_1, v_2, \dots, v_{\mathrm{K}} \in [\![0, v_{\max}]\!]$ be the list of the firing dates of the controlled execution, sorted in

a growing order. Formally we have: $\bigcup_{t\in\mathbb{T},k\in[\![1,K_t]\!]} u_k^t = \{v_1, v_2, \ldots, v_{\mathrm{K}}\} = \mathbb{V}$. At each date $v_k \in \mathbb{V}$, we build a step $\sigma_k = \sum_{j\in[\![1,\mathrm{N}]\!]} \alpha_j^k \cdot t_j$ such that $\sigma_k = \{t \in \mathbb{T} \mid \exists r \in [\![1, \mathrm{K}_t]\!],\ u_r^t = v_k\}$.

From equation (2.1), we have obviously $\forall j \in [\![1, \mathrm{N}]\!], \alpha_j^k \in \{0, 1\}$, thus the *step* (σ_k) can be used to build timed steps $\psi_k = (\sigma_k, v_k)$ satisfying the *non-reentrance condition* of definition (2.9).

We will show that there exists K states $s_1, s_2, \ldots, s_{\mathrm{K}} \in \mathscr{S}(R, d)$ such that $\forall k \in [\![1, \mathrm{K}]\!], s_{k-1}[\psi_k\rangle s_k$, and that these states are precisely the *instantaneous states* reached by the CE at the respective dates $v_1, v_2, \ldots, v_{\mathrm{K}}$.

Thus, a sequence of timed step firings can be got from a feasible controlled execution.

- $(\Leftarrow)$: Let $\Psi = \Big(\psi_1 = (\sigma_1, v_1),\ \ldots,\ \psi_{\mathrm{K}} = (\sigma_{\mathrm{K}}, v_{\mathrm{K}}),\ \psi_{\max} = (\overrightarrow{0_{\mathrm{N}}}, v_{\max})\Big)$ be a fireable timed step firing sequence leading from s_0 to s_f. Let $(s_k = (\overrightarrow{s_{m_k}}, \overrightarrow{s_{r_k}}))_{k\in[\![1,\mathrm{K}]\!]}$ be the states reached by the timed step firing sequence such that $s_0[\psi_1\rangle s_1 = [\ldots\rangle s_{\mathrm{K}-1}[\psi_{\mathrm{K}}\rangle s_{\mathrm{K}}[\psi_{\max}\rangle s_f$. Let $\mathbb{V} = \{v_1, v_2, \ldots, v_{\mathrm{K}}, v_{\max}\}$. Let $(u_k^t)_{t\in\mathbb{T},k\in[\![1,\mathrm{K}_t]\!]}$ be a controlled execution defined by:

  - $\forall t \in \mathbb{T}, \mathrm{K}_t = card(\{k \in [\![1, \mathrm{K}]\!], t \in \sigma_k\})$;
  - $\forall t \in \mathbb{T}$, $(u_k^t) = v_1^t, v_2^t, \ldots, v_{\mathrm{K}_t}^t \in \mathbb{V}$ is the increasing sequence of the firing dates of t in the timed step firing sequence: $\forall r \in [\![1, \mathrm{K}_t]\!], \exists! k \in [\![1, \mathrm{K}]\!], v_k = v_r^t$ and $t \in \sigma_k$.

  We will show that (u_k^t) is a feasible controlled execution leading to s_f from s_0 at date $v_{\max}$.

Thus, both *controlled executions* and timed step firing sequences are equivalent with each other. Therefore a corollary is given as follow.

Corollary 2.13 (Capture the Behavior of TPN by timed step firing sequences). *Let (R, d) be a TPN with its initial state s_0 given at date $v_0 = 0$.*

Every Controlled Execution can be represented as a sequence of timed steps. Thus, every state s_f reachable from s_0 can be reached by a sequence of timed steps firings. The length of such sequence depends on the final marking to reach s_f.

These results are quite obvious since both formulations follow the underlying semantics of ordinary PN, and firing sets T_k^f and T_k^s used in definition (2.10) are directly related to vectors $\overrightarrow{N(v)}$ and $\overrightarrow{F(v)}$ used in definition (2.4).

Indeed, the relations between firing sets and characteristic vectors can be expressed as follow equations:

$$T_{k+1}^f = \left\{ t \in \mathbb{T} : \left.\overrightarrow{F(v_k)}\right|_t = \left.\overrightarrow{N(v_k)}\right|_t - 1 \ \wedge \ \left.\overrightarrow{F(v_{k+1})}\right|_t = \left.\overrightarrow{N(v_k)}\right|_t \right\} \tag{2.12}$$

$$T_{k+1}^s = \left\{ t \in \mathbb{T} : \left.\overrightarrow{F(v_k)}\right|_t = \left.\overrightarrow{N(v_k)}\right|_t - 1 \ \wedge \ \left.\overrightarrow{F(v_{k+1})}\right|_t = \left.\overrightarrow{F(v_k)}\right|_t \right\} \tag{2.13}$$

The equations (2.12) and (2.13) can be derived from definition (2.10).

Let us recall firing sets of *Finished* transitions and *Still firing* in definition (2.10).

$$\forall t \in T_{k+1}^f, \quad \exists v_i, i \in [\![1,k]\!] \text{ s.t.} \quad v_k < v_i + d(t) \leqslant v_{k+1} \tag{2.14}$$

$$\forall t \in T_{k+1}^s, \quad \exists v_i, i \in [\![1,k]\!] \text{ s.t.} \quad v_k < v_{k+1} < v_i + d(t) \tag{2.15}$$

Suppose that v_i is the $(n+1)^{th}$ firing date in controlled executions, we can get the situation as:

$$\begin{cases} \left.\overrightarrow{N(v_i)}\right|_t = n+1 \\ \left.\overrightarrow{F(v_i)}\right|_t = n \\ \overrightarrow{s_{r_i}}(t) = d(t) \end{cases} \tag{2.16}$$

Let us consider the set of *Finished* transitions:

$$\begin{cases} v_i \leqslant v_k \\ v_k < v_i + d(t) \leqslant v_{k+1} \\ t \text{ is no reentrance} \end{cases} \Rightarrow \begin{cases} (n+1)^{th} \text{ firing is not finished at date } v_k \\ (n+1)^{th} \text{ firing is finished at date } v_{k+1} \end{cases}$$

$$\Rightarrow \begin{cases} \overrightarrow{N(v_k)}\Big|_t = \overrightarrow{N(v_i)}\Big|_t = n+1 \\ \overrightarrow{F(v_k)}\Big|_t = \overrightarrow{F(v_i)}\Big|_t = n \\ \overrightarrow{F(v_{k+1})}\Big|_t = n+1 \end{cases} \Rightarrow \begin{cases} \overrightarrow{F(v_{k+1})}\Big|_t = \overrightarrow{N(v_k)}\Big|_t \\ \overrightarrow{F(v_k)}\Big|_t = \overrightarrow{N(v_k)}\Big|_t - 1 \end{cases}$$

Let us consider the set of *Still firing* transitions:

$$\begin{cases} v_i \leqslant v_k \\ v_k < v_{k+1} < v_i + d(t) \\ t \text{ is no reentrance} \end{cases} \Rightarrow \begin{cases} (n+1)^{th} \text{ firing is not finished at date } v_k \\ (n+1)^{th} \text{ firing is not finished at date } v_{k+1} \end{cases}$$

$$\Rightarrow \begin{cases} \overrightarrow{N(v_k)}\Big|_t = \overrightarrow{N(v_i)}\Big|_t = n+1 \\ \overrightarrow{F(v_k)}\Big|_t = \overrightarrow{F(v_i)}\Big|_t = n \\ \overrightarrow{F(v_{k+1})}\Big|_t = n \end{cases} \Rightarrow \begin{cases} \overrightarrow{F(v_k)}\Big|_t = \overrightarrow{N(v_k)}\Big|_t - 1 \\ \overrightarrow{F(v_{k+1})}\Big|_t = \overrightarrow{F(v_k)}\Big|_t \end{cases}$$

From the previous equations, one must note that the firings sets at date $k+1$ can be *fully described* using the situation given at date k. Therefore, we rewrite the T^s_{k+1} and T^f_{k+1} using only $\overrightarrow{s_{r_k}}$ and Δ_{v_k} as follow:

$$T^f_{k+1} = \{t \in \mathbb{T}, \overrightarrow{s_{r_k}}(t) > 0 \wedge \quad \overrightarrow{s_{r_k}}(t) - \Delta_{v_k} \leqslant 0\} \tag{2.17}$$

$$T^s_{k+1} = \{t \in \mathbb{T}, \overrightarrow{s_{r_k}}(t) - \Delta_{v_k} > 0\} \tag{2.18}$$

For the set of *Finished* transitions:

$$\begin{cases} v_i \leqslant v_k \\ v_k < v_{k+1} < v_i + d(t) \\ t \text{ is no reentrance} \end{cases} \Rightarrow \begin{cases} \overrightarrow{s_{r_k}}(t) = d(t) - (v_k - v_i) \\ \overrightarrow{s_{r_k}}(t) - \Delta_k = d(t) - (v_k - v_i) - (v_{k+1} - v_k) \end{cases}$$

$$\Rightarrow \begin{cases} \overrightarrow{s_{r_k}}(t) = v_i + d(t) - v_k > 0 \\ \overrightarrow{s_{r_k}}(t) - \Delta_k = v_i + d(t) - v_{k+1} \leqslant 0 \end{cases}$$

For the set of *Still Firing* transitions:

$$\begin{cases} v_i \leqslant v_k \\ v_k < v_i + d(t) \leqslant v_{k+1} \\ t \text{ is no reentrance} \end{cases} \Rightarrow \begin{cases} \overrightarrow{s_{r_k}}(t) = d(t) - (v_k - v_i) \\ \overrightarrow{s_{r_k}}(t) - \Delta_k = d(t) - (v_k - v_i) - (v_{k+1} - v_k) \end{cases}$$

$$\Rightarrow \begin{cases} \overrightarrow{s_{r_k}}(t) = v_i + d(t) - v_k > 0 \\ \overrightarrow{s_{r_k}}(t) - \Delta_k = v_i + d(t) - v_{k+1} > 0 \end{cases} \overset{\Delta_k > 0}{\Rightarrow} \overrightarrow{s_{r_k}}(t) - \Delta_k > 0$$

Since equations (2.17) and (2.18) are non-linear equations, both *conditional constraint* and *linearization* will be applied for formulating them into mathematical model.

From the definition (2.11) and equations (2.7 - 2.10), we can conclude that the state $s_{k+1} = (\overrightarrow{s_{m_{k+1}}}, \overrightarrow{s_{r_{k+1}}})$ can be fully expressed by information from *previous step* $s_k = (\overrightarrow{s_{m_k}}, \overrightarrow{s_{r_k}})$ and variables denoting next step: $\overrightarrow{\sigma_{k+1}}$ and Δ_{v_k}. By simplifying expressions of the behavior of the TPN, and making it fully *incremental*, one can progressively increase the number of timed steps used in the formulation without redefining the whole set of constraints.

Following these propositions, it is sufficient to search for timed steps sequences to solve the reachability problem in TPN.

Corollary 2.14 (TPN Reachability Problem using Timed Steps Firings). *Let (R, d) be a TPN with its initial state $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_N}\right)$ given at date 0. Let $s_f = \left(\overrightarrow{s_{m_f}}, \overrightarrow{0_N}\right)$ be a target state. The reachability problem of TPNs consists*

in finding a firable sequence of timed steps $\psi_k = (\overrightarrow{\sigma_k}, v_k), k \in [\![1, \mathrm{K}]\!]$ *such that* $s_{k-1}[\psi_k\rangle s_k, s_{\mathrm{K}}[\psi_{\max}\rangle s_f$.

The advantage of using timed steps is that they allow to *reduce* the number of firings in our model – and then the number of variables – while keeping an equivalence with the initial properties. Thus it is not a *modification of the semantics* of TPNs, but only a way to *capture independent transitions*. Of course, this reduction does not systematically holds, since it is easy to construct a TPNs where only one transition can be fired at a time. Thus, in the worst case, timed step firing sequences formulation may not bring any improvement as far as the *number* of firings used are concerned. However, this is a quite uncommon situation since this should mean that the PN does not show any *parallelism*.

The contribution of proving the equivalence between Controlled Executions and Timed Step Firings and giving corollaries are published in our paper [HBYT13].

2.3 Incremental Search

In last sections, we have shown the interest of using timed steps to formulate the reachability problem using *incremental* method. The problem is expressed as a search for instanciations of integer variables constrained by the formulation of linear equations (2.7 - 2.10) and non-linear equations (2.17, 2.18). This formulation allows us to use the paradigm of *constraint programming* to solve the reachability problem.

However, the *initial definition* of the reachability problem is not well adapted to the kind of formulation we propose to use, since definition (2.7) does not make any assumption concerning the number of timed steps needed to solve the reachability problem.

Therefore, we propose several incremental search algorithms to improve the efficiency of the search process, as shown in figure (2.2). Before explaining this figure, we define two *sub-problems* associated with the original reachability problem introduced before, which can be conveniently solved using the characterization of corollary (2.14) in a constraint programming framework.

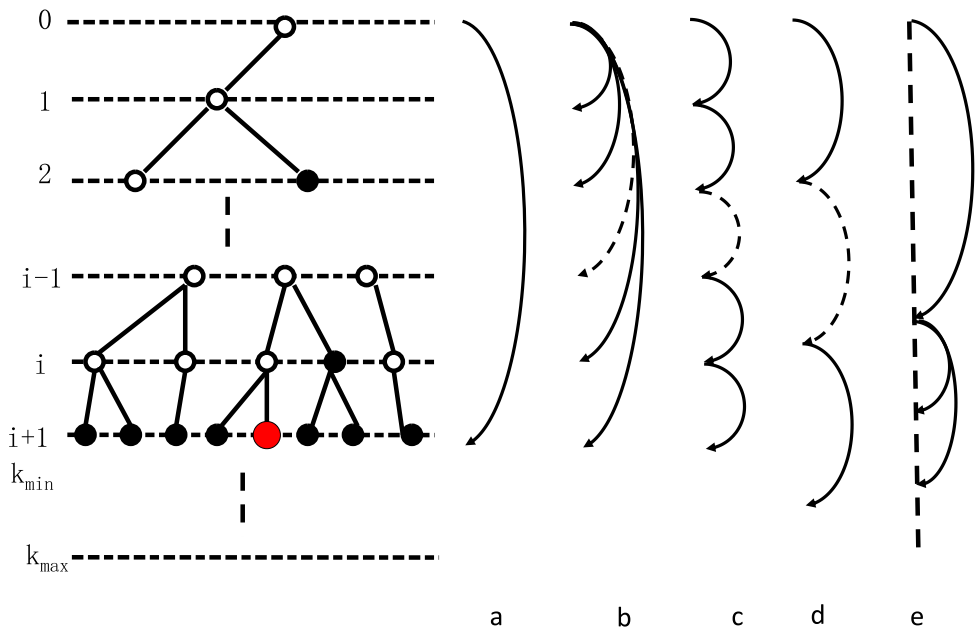


Figure 2.2: Incremental Search Algorithms

Definition 2.15 (Fixed Depth TPN Reachability Problem). *Let (R, d) be a TPN with its initial state $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ given at date 0. Let $s_f = \left(\overrightarrow{s_{m_f}}, \overrightarrow{0_{\mathrm{N}}}\right)$ be a target state.*

$\mathsf{TP}_1(k)$ *Find a timed step firing allowing to reach the state s_f from the state s_0 in at most k timed steps.*

Definition 2.16 (Shortest Length Reachability Problem). *Let (R, d) be a TPN with its initial state $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ given at date 0. Let $s_f = \left(\overrightarrow{s_{m_f}}, \overrightarrow{0_{\mathrm{N}}}\right)$ be a target state.*

TP_2 *Find the minimal length, denoted by $\mathrm{K_{min}}$, of a sequence of timed steps allowing to reach the state s_f from the state s_0.*

Of course, each of these sub-problems is *directly linked* to the initial one defined before, and each allows to solve a different kind of TPN reachability analysis. For instance, the first formulation $\mathsf{TP}_1(k)$ is highly useful for *model-checking* since it can serve to define the reachability graph under a bounded depth k. On the other hand, the second formulation TP_2 is designed to deal with

performance analysis since it returns a timed firing sequence that *maximizes the parallelism* of the system. It can also give an helpful bound for the definition of *additional heuristics*. Finally, since it is clear that the complexity of the problem grows (w.r.t. number of variables and constraints) as the length K of the sequence of steps used increases, it seems also quite reasonable to search for the smallest value of the parameter K from which a solution exists.

2.3.1 Naive Algorithm

In this thesis, we propose two kind of naive algorithms as figures (2.2b) and (2.2c). These algorithms are fed with a *bound* K_{max} on what we call the *Completeness Threshold or Sequential Depth* [BCCZ99]. Once chosen this value, the procedure generates iteratively a sequence of mathematical models of increasing size, and search for solutions in the corresponding search spaces. If there is no solution in less than K_{max} steps, the algorithm stops.

Naive Algorithm 1

Figure (2.2b) is described in figure (2.3).

1: $k \leftarrow 0$
2: **DO**
3: $k \leftarrow k + 1$
4: Generate $\mathsf{CP}(k)$, an incremental model for the problem $\mathsf{TP}_1(k)$ (which corresponds to characterization of corollary (2.14) with k timed steps).
5: Solve the model $\mathsf{CP}(k)$ (e.g. using *Ilog* solver). Let $\overrightarrow{X}_{i[\![1,k]\!]}$ be a solution of $\mathsf{CP}(k)$ if it exists.
6: **IF** ($\mathsf{CP}(k)$ *has a solution*), **RETURN** $\overrightarrow{X}_{i[\![1,k]\!]}$
7: **WHILE** ($\mathsf{CP}(k)$ *is infeasible*) **AND** ($k \leqslant \mathrm{K}_{max}$)

Figure 2.3: Naive Search Algorithm 1

In figure (2.2b) and (2.3), the algorithm will start a new search $CP(k), k = i$ from the root node when there is no solution at the search $CP(k), k = i - 1$. The advantage is that it does not need to record information from the search

$CP(k), k = i - 1$, since timed steps will represent the behavior of TPNs with length of k. But the disadvantage is apparent. It should consider nodes belonging to lower depth each time.

The promising aspect of the naive search algorithm 1 is that if the k_{min} is first given, as shown in figure (2.2a), it can avoid to explore many branches at lower depth of the search tree. Especially, in the best situation, the search strategies can explore the search tree without meeting deadends.

However, the K_{min} is hardly to be obtained as the reachability problem.

Naive Algorithm 2

Figure (2.2c) is described in figure (2.4).

1: $k \leftarrow 0$
2: **DO**
3: $k \leftarrow k + 1$
4: Generate $\mathsf{CP}(k)$, an incremental model for the set of problems $\{\mathsf{TP}_1(1)\}$, in which each state in the set of reachable states $\{s_{k-1}\}$ is initialed as s_0.
5: Solve $\mathsf{CP}(k)$. Let $\overrightarrow{X}_{i[\![1,k]\!]}$ be a solution of $\mathsf{CP}(k)$ if it exists.
6: **IF** ($\mathsf{CP}(k)$ *has a solution*), **RETURN** $\overrightarrow{X}_{i[\![1,k]\!]}$
7: **ELSE RETURN** the set of reachable states $\{s_k\}$
8: **WHILE** ($\mathsf{CP}(k)$ *is infeasible*) **AND** ($k \leqslant \mathrm{K}_{max}$)

Figure 2.4: Naive Search Algorithm 2

The advantage of the naive algorithm 2 is that it generates a new $\mathsf{CP}(k)$ based on all reachable states at depth $k-1$, which will not search for lower depth again like the naive algorithm 1.

However, it will lose benefits of timed steps and has to record reachable states at each depth. Meanwhile, search strategies are not needed, since all reachable states are explored. These disadvantage will lead our approaches to face the same problem as BDD, which will need too much memory when applying to large scale systems.

2.3.2 Jump Search

During the formulation of an incremental model at step 4 in figures (2.3) and (2.4), one should take care of the *domain of variables* representing timed steps. Indeed: the definitions of timed step and timed step firings do not forbid *empty timed steps* leaving the markings unchanged. By considering that empty timed steps are valid in our formulations, we get the following result.

Proposition 2.17 (Satisfaction Monotony). *Let $k \in \mathbb{N}$. If the problem $\mathsf{TP}_1(k)$ is feasible, then for any integer $k' \geqslant k$, the problem $\mathsf{TP}_1(k')$ is also feasible.*

Proof. It is easy to construct a feasible solution for $\mathsf{TP}_1(\mathrm{K}')$ from a feasible solution of $\mathsf{TP}_1(k)$ by adding empty steps.

□

Note the same result would be true when dealing with the family of incremental models $\mathsf{CP}(k)$: if there exists $k \in \mathbb{N}$ s.t. $\mathsf{CP}(k)$ admits a solution, any model $\mathsf{CP}(k')$ with $k' \geqslant k$ would be feasible too. This property motivates the jump search techniques proposed in the next paragraph.

From proposition (2.17) and the definition of parameter $K_{\min}$, we get:

$$\begin{cases} \forall k < K_{\min} & \mathsf{TP}_1(k) \text{ is infeasible} \\ \forall k \geqslant K_{\min} & \mathsf{TP}_1(k) \text{ is feasible} \end{cases} \tag{2.19}$$

This property can help us to define new iterative techniques, for example it shows that it is not necessary to solve all the problems $\mathsf{TP}_1(k)$ for $k \leqslant \mathrm{K}_{\min}$, like in the naive search described before.

Of course, as said before, we must keep using an *incremental procedure* in order to avoid the use of large models if they are not needed. We propose finally some techniques based on *jumps* over the values of the search depth. These techniques allow to *decrease* the number of iterations needed, thus improving the search efficiency. Several jump strategies are possible. We describe briefly some elementary ones.

- **Forward jump search.** The first family *continously increases* the value of the search depth. We can distinguish two main politics, depending on how the amplitude of jumps are defined.

- **Fixed amplitude.** Its value k_{jump} must be chosen in order to obtain a high exploration speed while minimizing the possible redundant steps, as presented in figure (2.2d).

  In figure (2.2), algorithm d is similar to algorithm c. It does not search for lower depth repeatedly and has to record reachable states at depth $n{*}k_{jump}$, where n represents times of launching new incremental search. In fact, algorithm c can be seen as $k_{jump} = 1$. Thus, the algorithm d will use less memory than the algorithm c. Meanwhile, techniques of timed steps and search strategies can help to enforce the process of exploring the search tree.

- **Dynamic amplitude.** This second strategy uses *variable amplitudes*. Increasing amplitudes should be used for small values of k, and decreasing ones when the exploration becomes more difficult. This kind of behavior is less easily quantifiable.

This politic can lead to overtake $\mathrm{K}_{\min}$ when a solution is found. In this case, it is not anymore possible to answer precisely the problem TP_2, since we do not get the exact value of $\mathrm{K}_{\min}$. To compensate this lack of information, one can use a dichotomic search.

- **Dichotomic search.** This kind of procedure needs to know a maximal bound for K_{max}. Its value is given by a previous successful execution of the forward jump search. For example, if K_{max} is known as in figure (2.2e), the algorithm e will dichotomize K_{max} and solve the problem $\mathsf{TP}_1(K_{max}/2)$ first. If there is no solution found, it will search between depths $K_{max}/2$ and $3 * K_{max}/4$, etc.

The main interest of jump search is that it allows to reduce some branches in the search tree. Since we do not know the number of steps needed to find a solution if it exists, the use of such a technique allows us, when it is possible, not to have to develop the entire set of formulations of length lower than $\mathrm{K}_{\min}$.

Finally, it must be said that procedures described in figures (2.3) and (2.4) are only *semi-complete* algorithms. Indeed, in the context of *unbounded* TPNs, the value of $\mathrm{K}_{\max}$ is set arbitrarily, as we do not know any information on the number

of steps needed to find a possible solution. Thus, if *no solution* is obtained *before* the value of K_{max} has been reached, one *cannot conclude* on the reachability property.

To the contrary, when dealing with bounded TPNs, it is possible to set K_{max} to the value of the *Completeness Threshold or Sequential Depth* of the net, a parameter that has been defined in [BHY04] and which *guarantee* the *complete exploration* of the reachability graph. Using this parameter as search depth, it is *always possible* to conclude when the algorithm stops.

In this thesis, we mainly use the algorithm like presented in figures (2.2b) and (2.3). The integer number k is first given based on the TPN model. The motivation to use this algorithm is to adapt advantages of timed steps and search strategies, which will lead solver to the desired marking without exploring too much in the search tree and without recording intermediate states.

2.4 Conclusion

In this chapter, we have introduced the fundamental model for analyzing TPNs called incremental model for TPNs. All developed methods in the following chapters will be derived from this incremental model.

First, the formal definition of TPNs and their terminology are introduced based on [CC88]. In addition to ordinary PNs, the *Residual Duration vector* is defined by expressing the missing tokens during a timed transition firing. Then *Controlled Execution* introduced in [CC88] are defined for presenting the TPNs instantaneous state and reachability problem.

Second, the incremental approach is introduced based on an *implicit traversal* of underlying reachability graph of TPNs, which does not need its whole construction. This is done by considering a unique sequence of timed steps growing incrementally to represent exactly the total behavior of TPNs. The equivalence between timed step firing and Controlled Executions is formally proved to express the correctness of our formula. Then the reachability problem of TPN can be expressed as finding a reachable incremental timed steps firings.

The advantage of incremental model is that it can express any states by timed steps, instead of representing all states explicitly. Our model is built as general as

possible since we do not make assumptions about the firing policy, contrariwise to other classical approaches dealing with the same issue.

These advantages can be efficient for search for solutions, since it allows to *reduce* the number of firings in the system - and then the number of variables - while keeping an equivalence with the initial properties. Meanwhile, search strategies can be applied to search for solutions more quickly.

Finally, several incremental search algorithms are introduced to solve reachability problems of TPNs. The most important factor is considered: the depth of K, which lead to us define two sub-problem: *fixed depth reachability problem* and *shortest length reachability problem*. Then, *naive algorithm* and *jump search* techniques are introduced to help us to develop more efficient search algorithm.

In this thesis, the incremental search algorithm presented in figures (2.2b) and (2.3) is applied, since it can adapt advantages of timed steps and search strategies.

In next chapter, we propose to use CP to formulate the incremental approach as mathematical formulations. In CP, several techniques can be used to continue reducing the influence of combinatorial explosion, such as adding constraints, reducing domains of variables and developing search strategies. Meanwhile *conditional constraint* can be used to express the non-linear equations (2.17) and (2.18) more naturally.

Chapter 3

Analysis of Timed Petri Nets using Constraint Programming

The objective of this chapter is to formulate and implement our incremental model with ***Constraint Programming*** (CP), then to find one or all solutions using search strategies in CP. With several techniques of CP, such as adding constraints, reducing domains of variables and developing search strategies, the influence of *combinatorial explosion* can be reduced.

In fact, CP is mainly used to solve a *constraint satisfaction problem* (CSP), which can be defined using *variables* and their domains, and *constraints* that represent the relations between these variables. An incremental model can be easily expressed as a CSP, especially the `non-linear` equations (2.17) and (2.18) can be formulated as *conditional constraints* in a natural way. The CP mainly uses two stages to solve a CSP: ***Modeling*** and ***Solving***. In the *Modeling* phase, CP can reduce the *search space* formed by variables through adding some improvements, for instance, reduce domains of variables domains and dynamically add constraints. Such improvements will reduce the influence of combinatorial explosion at the same time as reducing search space. In the *Solving* phase, some developed search strategies can find a solution by traversing the *search tree* more efficiently, since irrelevant nodes or elements can be eliminated during the search. Such search process can reduce the influence of combinatorial explosion in a controlled way. At the best case, CP can find one solution with no fail nodes which means without expanding any other irrelevant states in the reachability graph.

We will also highlight the flexibility of CP for expressing some specific issues in TPN model. For example, the `Token Identification` that will be introduced in chapter (4) can be easily expressed by adding some new variables and constraints in the *Model* phase, and then will be solved combined with previous model in the *Model* phase. Some new efficient search strategies can also be developed based on the Token Identification.

In the first section, a tutorial introduction of CP is given to express the behavior. A CSP is a basic problem in the real world. Generally speaking, CP solving contains two stages: *Modeling* and *Solving*. The *Modeling* stage describes the system and relations in them using variables and constraints. When dealing with manufacturing systems in chapter (4), many issues can be expressed by adding variables and constraints, which will lead to more realistic benchmarks. The *Solving* stage searches for solutions using different search strategies combined with constraint propagation. The process of solving is very general in CP world. The solver will first generate a *search tree* with only the root node. Then, it will branch the search tree by selecting a variable with different *variable ordering* heuristics. After selecting a variable to be labeled, it will choose the *value ordering* to select the leaves for *constraint propagation*. Finally, when it meets a *deadend*, it will *backtrack* to an open node to continue the search until a solution is found. The common algorithm in this process is using *depth-first* backtrack and *arc consistency* as constraint propagation.

In the second section, we *model* the incremental model. The physical sense elements in TPNs are defined as variables, like places, transitions, markings, residual duration vectors, etc. And the structure and relations between these elements are expressed as constraints, like incidence matrix, state equations, etc. Especially, the non-linear equations are modeled by conditional constraints. Then the incremental model is translated into constraint programming model, which can be implemented to solve the reachability problem. To solve the reachability problem more efficiently, modeling improvements are given to reduce the search space.

In the third section, we introduce different search strategies with an example. *Generic* strategies are given by CP techniques and *dedicated* strategies are developed based on the knowledge of the TPN behavior and the meaning of variables.

In addition, many techniques can be applied to improve the efficiency of searching. Especially, an objective can be added to search for an optimal solution to handle scheduling issues.

At the last section, a linearization technique is applied to improve the efficiency of expressing non-linear part by conditional constraint. The linearization constraints can also be seen as redundancy constraints working with conditional constraints.

Finally, some benchmarks are given to explain all considerations above.

3.1 Constraint Programming Tutorial

Constraint Programming is the study of computational systems based on constraints. The main idea of constraint programming is to solve problems by combining constraints (conditions, properties or requirements) about the problem area and, subsequently finding a solution satisfying all of the constraints. The earliest ideas leading to CP can be found in Artificial Intelligence (AI), dating back to sixties and seventies. Logic Programming (LP) can also be noted to be just a particular kind of CP. CP techniques are used to solve hard combinatorial problems and are very competitive with techniques from Operations Research (OR). CP is now a mature field and has been successfully used for tackling a wide range of complex problems [Mic07].

3.1.1 CP Terminology

The term of Constraint Programming (CP) refers to the techniques that are used to represent and solve Constraint Satisfaction Problems and Constraint Optimisation Problems arising from Artificial Intelligence [He12]. This section gives a brief introduction and basic notation of CP. A large part of this section is written based on the books [Apt03] and [RBW06].

Constraint satisfaction problems, like most fields of artificial intelligence, can be separated into (overlapping) concerns of representation and reasoning. The former can be divided into generic and application-specific concerns, the latter

into search and inference. Classic definition of a Constraint Satisfaction Problem (CSP) can be expressed as follows:

Definition 3.1 (Constraint Satisfaction Problem). *A CSP is a triple* $P = \langle X, D, C\rangle$*, where* X *is an* n*-tuple of variables* $X = \langle x_1, x_2, \ldots, x_n\rangle$*,* D *is a corresponding* n*-tuple of domains* $D = \langle D_1, D_2, \ldots, D_n\rangle$*, such that* $x_i \in D_i$*,* C *is a* t*-tuple of constraints* $C = \langle C_1, C_2, \ldots, C_t\rangle$*. A constraint* C_j *is a pair* $\langle R_{S_j}, Sj\rangle$ *where* R_{S_j} *is a relation on the variables in* $S_j = scope(C_j)$*. In other words,* R_{S_j} *is defined as a subset of the Cartesian product of the domains of the variables in* S_j*.*

Definition 3.2 (Satisfaction). *For a CSP* $P = \langle X, D, C\rangle$*, if a* n*-tuple* $A = \langle a_1, a_2, \ldots, a_n\rangle$ *where* $a_i \in D_i$ *s.t. each* Cj *(*$C = \langle C_1, C_2, \ldots, C_t\rangle$*) is satisfied in that* R_{S_j} *holds on the projection of* A *onto the scope* S_j*,* A *is said to be a solution of this CSP.*

Briefly, when a CSP is given, CP searches values for all the variables satisfying all constraints. In addition, CP is also used to search optimal solutions. Generally, an optimal solution is given with respect to a certain criteria, and CP will traverse the search tree to ensure its optimality.

Definition 3.3 (Constraint Optimisation Problem). *A Constraint Optimisation Problem (COP) is a CSP(****X, D, C****) together with an objective function* $f : D_1 \times D_2 \times \ldots \times D_n \rightarrow \boldsymbol{R}$ *to be optimised. An optimal solution to a constraint optimisation problem is a solution to* P *that is optimal with respect to* f*. The objective function value is often represented by a variable* z*, together with maximizing* z *or minimizing* z *for maximization or a minimization problem, respectively*

In CP, the goal is to find a solution (or all solutions) to a given CSP, or an optimal solution (or all optimal solutions) to a given COP. To mention COP here, we want to give a brief introduction of CP techniques, in which COP is a very important part, and to tell the reader that our model can be easily applied for optimization problems in TPNs by adding objectives.

After representing a CSP, the domains D_i will form a *search space* for putative solutions Ω. Then two broad categeries of algorithms for solving CSPs will

be considered: inference and search. In inference techniques, *Constraint Propagation* can eliminate large subspaces from Ω using *Network Consistency*. *Search* systematically explores Ω, often eliminating subspaces with a single failure. The success of both strategies hinges on the simple fact that a CSP is conjunctive: to solve it, all of the constraints must be satisfied so that a local failure on a subset of variables rules out all putative solutions with the same projection onto those variables. These two basic strategies are usually combined in most applications [RBW06].

In summary, we can list a basic structure of CP program describing a CSP as the following:

- definition of variables and domains
- equations of constraints among variables
- definition of search strategies (which are executed with the constraint propagation)

3.1.1.1 Constraint Propagation

Constraint propagation removes (some) inconsistent values from their corresponding domains, based on the considerations on the individual constraints. By doing so, the search space can be significantly reduced. Hence, constraint propagation is essential to make constraint programming solvers efficient.

Let C be a constraint on the variables $X = \langle x_1, x_2, \ldots, x_n \rangle$ with respective domains $D = \langle D_1, D_2, \ldots, D_n \rangle$, such that $x_i \in D_i$. A *propagation algorithm* for C removes the values from $D = \langle D_1, D_2, \ldots, D_n \rangle$ that do not participate in a solution to C.

Let $P = \langle X, D, C \rangle$ be a CSP. We transform P into a smaller CSP P by repeatedly applying the propagation algorithms for all constraints in C until there is no more domain reduction. This process is called *constraint propagation*. When the process terminates, we say that each constraint, and the CSP, is locally consistent and that we have achieved a notion of *local consistency* on the constraints and the CSP. The term local consistency reflects that we do not obtain a globally consistent CSP, but a CSP in which each constraint is locally consistent.

Constraint propagation algorithms are the algorithms that achieve local consistency. In the literature, it is also called *filtering algorithm*. For keeping local consistency in the constraint propagation, there are many algorithms like node consistency, arc consistency, k-consistency, etc. The differences between them are combining different number of constraints for one variable. All kinds of consistency and a thorough description of the process of constraint propagation are given by [Apt03]

The most popular notion of local consistency is *arc consistency* that deals with binary constraints. Arc consistency is the basic propagation mechanism that is used in almost all solvers.

Definition 3.4 (Arc consistency). *Consider a binary constraint C on the variables x and y with a domain of D_x and D_y respectively, that is $C \subseteq D_x \times D_y$. We call C arc consistent if*

$$\forall a \in D_x, \exists b \in D_y, (a, b) \in C \tag{3.1}$$

$$\forall b \in D_y, \exists a \in D_x, (a, b) \in C \tag{3.2}$$

We say a CSP is arc consistent if all its binary constraints are arc consistent.

Constraint propagation is usually applied each time when a domain has been changed. Consequently, the propagation algorithm that we apply to make a CSP locally consistent should be as efficient as possible. However, a propagation algorithm does not need to remove all such values, as this may lead to an exponential running time due to the nature of some constraints (see the complexity analysis of arc consistency in [RBW06].

3.1.1.2 Search

After constraint propagation, we usually encounter three kinds of scenarios:

- the problem is inconsistent, which means no feasible solution exist;
- there is only one value in each domain of variable, which means we found the solution;

- there is more than one value in each variable's domain, which means we have to continue the search for a solution.

The solution process of CP uses a search tree, which is a particular rooted tree. The vertices of search trees are often referred to as nodes. The arcs of search trees are often referred to as branches. Further, if (u, v) is an arc of a search tree, we say that v is a direct descendant of u and u is the parent of v.

Definition 3.5 (Search tree). *Let P be a CSP. A search tree for P is a rooted tree such that:*

- *its nodes are CSPs*
- *its root is P_0*
- *if $\langle P_1, \ldots, P_m \rangle$ where $m > 0$ are all direct descendants of P_0, then the union of the solution sets of $\langle P_1, \ldots, P_m \rangle$ is equal to the solution set of P_0.*

We say that a node P of a search tree is at depth d if the length of the path from the root to P is d.

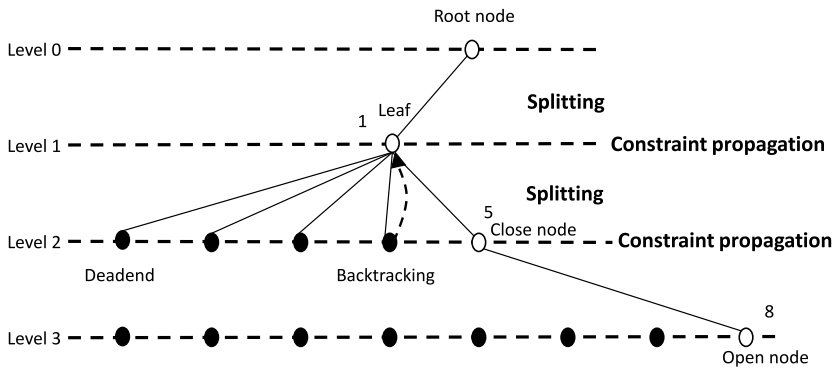


Figure 3.1: Expression of Search Tree

Definition (3.5) is a very general notion. In CP, a search tree is dynamically built by *splitting* (also called *branch* in the search tree) a CSP into smaller CSPs, until we reach a *failed* or a *solved* CSP. A CSP is split into smaller CSPs either

by splitting a constraint or by splitting the domain of a variable. For more information about splitting we refer readers to [Apt03].

At each node in the search tree we apply constraint propagation to the corresponding CSP. As a result, we may detect that the CSP is inconsistent, or we may reduce some domains of the CSP. In both cases fewer nodes need to be generated and traversed, so the propagation can speed up the solution process.

We say that a node P of a search tree is at level d if the length of the path from the root to P is d. As shown in figure (3.1), the *leaves* are branched from parent nodes. A *close node* means no more leaf can be expanded, and an *open node* is the beginning to a branch. The *deadend* means there is no solution with this leaf, so solver will *backtrack* to the parent node.

The constraint propagation and splitting are applied in an alternated fashion. The most common form of domain splitting consists in *enumerating* the domain of a variable. Informally, it consists of taking a variable, say x, and splitting its domain into singleton sets. Each such singleton set, say a, corresponds to a CSP in which the domain of the variable x is replaced by a. Another domain splitting consists of *binary choice points* in the domain of a variable, known as *binary tree*.

Variable and Value Ordering Heuristics

To split the domain of a variable, we first select a variable and then decide how to split its domain. This process is guided by variable and value ordering heuristics. Heuristic is defined as a *rule of thumb* based on domain knowledge from a particular application, which gives guidance in the solution of a problem. Variable and value ordering heuristics impose an ordering on the variables and values, respectively. The order in which variables and values are selected has a great impact on the search process.

A *variable ordering heuristic* imposes a partial order on the variables with non-singleton domains. An example is the *most constrained first* variable ordering heuristic. It orders the variables with respect to the number of their appearance in the constraints. A variable that appears the most often is ordered first. It is likely that changing the domains of such variables will cause more values to be removed by constraint propagation. Another variable ordering heuristic is the

smallest domain first heuristic. This heuristic orders the variables with respect to the size of their domains. A variable that has the smallest domain is ordered first. The advantages of this heuristic are that less nodes are needed to be generated in the search tree, and that inconsistent CSPs are detected earlier. In case two or more variables are incomparable, we can for example apply the lexicographic ordering to these variables and obtain a total order [GP10].

A *value ordering heuristic* induces a partial order on the domain of a variable. It orders the values in the domain according to a certain criterion. An example is the lexicographic value ordering heuristic, which orders the values with respect to the lexicographic ordering [GP10]. Another example is the random value ordering heuristic, which orders the variables randomly. In case a value ordering heuristic imposes a partial order on a domain, we can apply the lexicographic or random value ordering heuristic to incomparable values to create a total order. A value ordering heuristic is also referred to as a branching heuristic because it decides the order of the branches in the search tree.

Backtracking

After choosing variables ordering and value ordering, CP solver goes down in the search tree. When meeting a deadend, CP solver need to search in other branches. There are several algorithms to *traverse* the search tree. Complete or systematic algorithms come with a guarantee that a solution will be found if it exists, and can be also used to show that a CSP does not have a solution and to find an optimal solution. *Backtracking search algorithms* and *dynamic programming algorithms* are, in general, examples of complete algorithms. Incomplete, or non-systematic algorithms, cannot be used to show a CSP does not have a solution or to find a provably optimal solution. However, such algorithms are often effective at finding a solution if one exists and can be used to find an approximation to an optimal solution. *Local* or *stochastic search algorithms* are examples of incomplete algorithms [RBW06].

Backtracking search algorithm is the most popular one, since it works on only one solution at a time and thus needs only a polynomial amount of space. The most common search strategies is *Depth First Search*, which explores the search

tree in a *left-to-right* fashion. It searches by selecting the left leaf at each open node until the end of the search space is reached. If it does not find a solution, it backtracks and tries again.

In CP, constraint modelling, variable/value order heuristics and backtracking interact in the whole procedure of problem solving. None of these decisions can be made independently from the others. We use the term *search strategies* to express all the actions in the search process.

3.1.1.3 The IBM ILOG Solver

This thesis is the result of three years of research that was conducted and implemented based on IBM ILOG solver [IBM10].

IBM ILOG Solver 6.8 is a C++ CP solver. It is a C++ library for CP. It includes:

- predefined classes of variables;
- predefined classes of mathematic, symbolic, and global constraints, with associated one (or more) propagation algorithm, together with a mechanism to implement new constraints;
- predefined search algorithms, together with a mechanism to write user defined tree search methods

In the following, we will use *Solver* to replace IBM ILOG Solver.

Search in Solver is typically a tree search. It provides a set of control methods that allow users to implement their own search algorithms.

In this thesis, we mainly use the CP as a modeling tool, which can translate equations into constraints. Meanwhile, conditional constraints are very powerful for us to formulate non-linear equations. Then we solve the reachability problem with pre-defined and our developed search strategies.

In summary, the process of solving a CSP can be separated into two parts:
Part 1 ***Modeling***

- Definition of variables and their domains
- Expressing constraints among variables

Part 2 ***Solving***

- Definition of the objective function and its link with the problem decision variables
- Definition of the search strategy : variables and values ordering
- Definition of the search strategy : backtracking algorithm

Generally, a popular method to generate a search tree is using Depth-First backtracking search algorithm and keeping arc consistency during constraint propagation.

3.2 Modeling Incremental Model for Constraint Programming

In the previous chapter, we have first shown that timed step firings are sufficient to solve the reachability problem of TPN. Then the *incremental* model is built for searching for a timed step firing sequence *step by step*. In this section, we express the incremental model with CP techniques and show how a timed step firing can be expressed as *a constraint programming problem*.

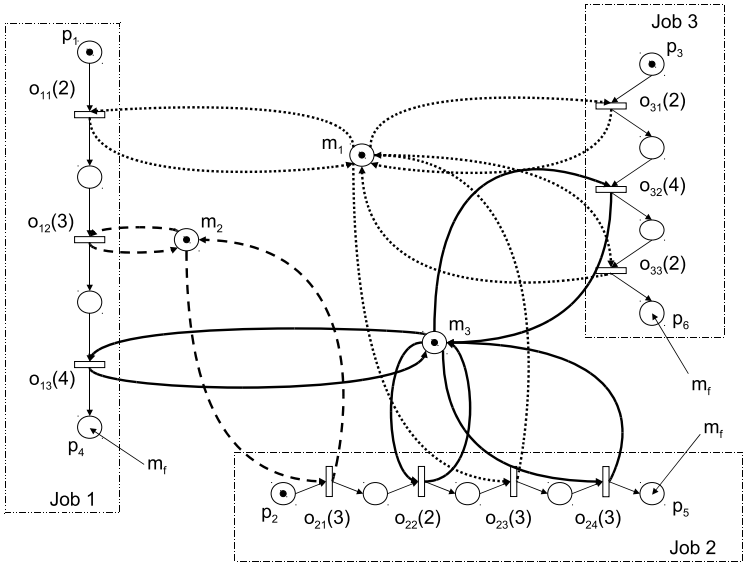
In the following, the modeling improvements are given for reducing search space, and different kind of strategies are shown for searching for solutions efficiently. Under these considerations, we propose to follow an example-driven tour for explaining these techniques and choose the way for more efficient solving.

Let us consider the example of figure (3.2), its corresponding incidence matrices and time durations of transitions are given.

We start by expressing our incremental model in a suitable way to be solved using CP.

3.2.1 Preliminary Modeling

As said in section (3.1), the *Modeling* process of CP for modeling a TPN is made of definition of variables and definition of constraints.



$$C^- = \begin{bmatrix} 1,0,0,0,0,0,0,0,0,0 \\ 0,1,0,0,0,0,0,0,0,0 \\ 0,0,1,0,0,0,0,0,0,0 \\ 0,0,0,0,0,0,0,0,0,0 \\ 0,0,0,1,0,0,0,0,0,0 \\ 0,0,0,0,1,0,0,0,0,0 \\ 0,0,0,0,0,1,0,0,0,0 \\ 0,0,0,0,0,0,1,0,0,0 \\ 0,0,0,0,0,0,0,0,0,0 \\ 0,0,0,0,0,0,0,1,0,0 \\ 0,0,0,0,0,0,0,0,1,0 \\ 0,0,0,0,0,0,0,0,0,1 \\ 0,0,0,0,0,0,0,0,0,0 \\ 1,0,0,0,0,1,0,1,0,1 \\ 0,1,0,1,0,0,0,0,0,0 \\ 0,0,1,0,1,0,1,0,1,0 \end{bmatrix} \qquad C^+ = \begin{bmatrix} 0,0,0,0,0,0,0,0,0,0 \\ 1,0,0,0,0,0,0,0,0,0 \\ 0,1,0,0,0,0,0,0,0,0 \\ 0,0,1,0,0,0,0,0,0,0 \\ 0,0,0,0,0,0,0,0,0,0 \\ 0,0,0,1,0,0,0,0,0,0 \\ 0,0,0,0,1,0,0,0,0,0 \\ 0,0,0,0,0,1,0,0,0,0 \\ 0,0,0,0,0,0,1,0,0,0 \\ 0,0,0,0,0,0,0,0,0,0 \\ 0,0,0,0,0,0,0,1,0,0 \\ 0,0,0,0,0,0,0,0,1,0 \\ 0,0,0,0,0,0,0,0,0,1 \\ 1,0,0,0,0,1,0,1,0,1 \\ 0,1,0,1,0,0,0,0,0,0 \\ 0,0,1,0,1,0,1,0,1,0 \end{bmatrix}$$

$\overrightarrow{d} = [2,3,4,3,2,3,3,2,4,2]$.
$\overrightarrow{s_{m_0}} = [2,0,0,0,2,0,0,0,0,2,0,0,0,1,1,1]$.
$\overrightarrow{s_{m_f}} = [0,0,0,2,0,0,0,0,2,0,0,0,2,1,1,1]$.
$\overrightarrow{s_{r_0}} = \overrightarrow{s_{r_f}} = \overrightarrow{0_N}$.

Figure 3.2: Modeling a Job Shop in TPNs

3.2.1.1 Variables Definition

In this section, we define variables used for TPNs and their domains. The main idea of incremental model is to use a sequence of timed step firings, as shown in table (2.3). So in each step, the following variables and their domains are considered:

- Step number (k), $k \in [\![0, \mathrm{K}]\!]$ indicates the number of steps;
- Step firing variables ($\overrightarrow{\sigma_k}$), $\forall j \in [\![1, N]\!], \overrightarrow{\sigma_k}(j) \in \{0, 1\}$ denotes if each transition in one step is fired or not;
- Firing Dates Variables (Δ_{v_k}), $\Delta_{v_k} = v_{k+1} - v_k, v_0 = 0, \Delta_{v_k} \in \mathbb{N}$ means the time elapse between two steps; Note that for $\Delta_{v_k}, k \in [\![0, \mathrm{K} - 1]\!]$.
- Intermediate Marking Variables ($\overrightarrow{s_{m_k}}$), $\forall i \in [\![1, M]\!]$, $\overrightarrow{s_{m_k}}(i) \in \mathbb{N}$ means the number of token in each place;
- Intermediate Residual Duration Variables ($\overrightarrow{s_{r_k}}$), $\forall j \in [1, N], \overrightarrow{s_{r_k}}(j) \in \mathbb{N}$ means the residual duration of each transition;
- Intermediate Finished Transition Variables ($\overrightarrow{T_k^f}$), $\forall j \in [1, N], \overrightarrow{T_k^f}(j) \in \{0, 1\}$ means if one transition is finished or not during time inteval $]v_{k-1}, v_k]$

Note that steps $k = 0$ and $k = K$ are respectively used to be instantiated by initial state and final state.

3.2.1.2 Constraints Expression

After defining variables, constraints should be expressed to describe the relations among variables.

Expression Of Marking Updates

Marking update equations (2.17 and 2.18) are represented below:

$$T_{k+1}^f = \{t \in \mathbb{T}, \overrightarrow{s_{r_k}}(t) > 0 \wedge \quad \overrightarrow{s_{r_k}}(t) - \Delta_{v_k} \leqslant 0\}$$
$$T_{k+1}^s = \{t \in \mathbb{T}, \overrightarrow{s_{r_k}}(t) - \Delta_{v_k} > 0\}$$

In this section, we propose to express the non-linear parts directly as conditional constraints equations. For considering tokens produced at date $k+1$, we add intermediate variables $TF_{(k+1)}$ to denote finishing transitions, using the following set of conditional constraints:

$$\forall j \in [\![1, \mathrm{N}]\!], \forall k \in [\![0, \mathrm{K}-1]\!]$$

$$\begin{cases} s_{r_{kj}} > 0 \\ \wedge \quad s_{r_{kj}} - \Delta_{v_k} \leqslant 0 \end{cases} \Rightarrow T^f_{(k+1)j} = 1 \tag{3.3}$$

$$\begin{cases} s_{r_{kj}} \leqslant 0 \\ \vee \quad s_{r_{kj}} - \Delta_k > 0 \end{cases} \Rightarrow T^f_{(k+1)j} = 0 \tag{3.4}$$

Equations (3.3) denote transitions j that were firing at date v_k and have finished at date v_{k+1}. Conversely, equations (3.4) denote transitions that were not firing at date v_k or that are still firing at date v_{k+1}.

The vector $\overrightarrow{T^f_{k+1}}$ is then used to express equation (2.9) in the following way:

$$\overrightarrow{s_{m_{k+1}}} = \overrightarrow{s_{m_k}} - C^- \cdot \overrightarrow{\sigma_{k+1}} + C^+ \cdot \overrightarrow{T^f_{k+1}} \tag{3.5}$$

Expression Of Residual Duration Vector Updates

To express equation (2.10), we do not explicitly make appear a vector characterizing the set T^s_{k+1}, but we use directly the values of T^s_k to check if previous firing transitions are still firing.

Since reentrance is forbidden, a transition t_j which was *already firing* at date v_k cannot start a new fire at step v_{k+1} if it has not finished its previous firing before. Thus, at date v_{k+1}, the corresponding component of the residual durations vector, $s_{r_{(k+1)j}}$, can have two values:

$$s_{r_{kj}} - \Delta_k > 0 \Rightarrow \begin{cases} s_{r_{(k+1)j}} = s_{r_{kj}} - \Delta_k \\ \sigma_{(k+1)j} = 0 \end{cases} \tag{3.6}$$

$$s_{r_{kj}} - \Delta_k \leqslant 0 \Rightarrow s_{r_{(k+1)j}} = d(j) \cdot \sigma_{(k+1)j} \tag{3.7}$$

Equation (3.6) means that t_j is still firing and cannot be fired anymore, so $s_{r_{(k+1)j}}$ is equal to the residual time to elapse. Following equation (3.7), if t_j is

not firing (or has finished) at date v_{k+1}, $s_{r_{(k+1)j}}$ is decided by the new step $\sigma_{(k+1)j}$. Equations (3.6) and (3.7) are used to express equation (2.10) in the final model.

Expression of Fireability Conditions

The monoserver semantics enforced by equation (2.7) is already guaranteed by the second part of equation (3.6).

In the same way, the expression of fireability condition (2.8) is a consequence of equation (3.5), since variables of vector $\overrightarrow{s_{m_{k+1}}}$ are constrained to be positive.

3.2.2 Modeling Improvements

In order to reduce the search space and improve the efficiency of CP, we reduce the domain of variables and add more constraints to the CP Model, according to some observations made on the considered TPNs.

Firing Policy Since all transitions have a finite duration, it is not necessary to *wait* after the firing of $\overrightarrow{\sigma_k}$ at date v_k more than the max residual duration $\overrightarrow{s_{r_k}}$, denoted as $max(\overrightarrow{s_{r_k}})$, to fire the next step $\overrightarrow{\sigma_{k+1}}$. Indeed: we are sure that all transitions previously active at date v_k have finished, thus all tokens needed to fire $\overrightarrow{\sigma_{k+1}}$ are available at date $v_k + max(\overrightarrow{s_{r_k}})$. To define the domains of Δ_{v_k} and $s_{r_{kj}}$, they will be no more than the max duration $D_{max} = max(\overrightarrow{d})$ of transitions, which is said as the longest duration. For each transition j, its residual duration will be smaller than its own duration d_j. Such considerations lead to

$$\forall k \in [\![0, \mathrm{K}]\!], \forall j \in [\![1, \mathrm{N}]\!],\ s_{r_{kj}} \in \{0, D_{max}\} \tag{3.8}$$
$$\forall k \in [\![0, \mathrm{K}]\!], \forall j \in [\![1, \mathrm{N}]\!],\ s_{r_{kj}} \leqslant\ d_j \tag{3.9}$$
$$\forall k \in [\![0, \mathrm{K}-1]\!],\ \Delta_{v_k} \in \{0, D_{max}\} \tag{3.10}$$
$$\forall k \in [\![0, \mathrm{K}-1]\!],\ \Delta_{v_k} \leqslant max(\overrightarrow{s_{r_k}}) \tag{3.11}$$

Note that equations (3.8 and 3.10) are used to define variables $\overrightarrow{s_{r_k}}$ and Δ_{v_k}. Equations (3.9 and 3.11) are used to define constraints on $\overrightarrow{s_{r_k}}$ and Δ_{v_k}.

In order to reduce the search space, we forbid the firing of *empty* steps, i.e. steps containing no transition firing. This avoid to find solution built

from another one by simply adding an empty step. Only the last steps $\overrightarrow{\sigma_{\mathrm{K}}}$ is allowed to be empty, in order to be able to consider the date K for which the final marking is reached but no transition is still active. Note this implies to choose the total number of steps accurately since if one choose more steps than the number needed to reach the final marking, no solution can be found. To the contrary, empty steps allow to use dichotomic search to find the minimal number of steps needed to reach the final marking.

In order to avoid solutions built one from another by splitting steps into individual transition firings, we forbid the elapsed time Δ_k between steps $\overrightarrow{\sigma_k}$ and $\overrightarrow{\sigma_{k+1}}$ to be equal to 0, except the first time elapse Δ_0. Thus, if a solution exists where two transitions t_i, t_j must be fired simultaneously at date v_k, they will be fired in an unique step $\overrightarrow{\sigma_k} = \overrightarrow{e_{t_i}} + \overrightarrow{e_{t_j}}$ and not using two steps $\overrightarrow{\sigma_k} = \overrightarrow{e_{t_i}}$ and $\overrightarrow{\sigma_{k+1}} = \overrightarrow{e_{t_j}}$ with $\Delta_k = 0$.

Such improvements reduce the search space by deleting solutions that would be found anyway. The corresponding constraints are given in equations (3.12) and (3.13).

$$\forall k \in [\![0, \mathrm{K}-1]\!], \sum_{j=0}^{\mathrm{N}-1} \sigma_{kj} \neq 0 \tag{3.12}$$

$$\forall k \in [\![1, \mathrm{K}-1]\!], \Delta_{v_k} \neq 0 \tag{3.13}$$

Structural Properties Since PNs express the behavior of physical systems, they have generally an infinite behavior, made of cyclic executions, but their number of states is *bounded*. Thus, it is possible to compute an upper bound of the marking of each place, allowing to reduce the domain of variables $\overrightarrow{s_{m_k}}$ in the considered model, as shown in equation (3.14). Such bounds can be computed using P-flows of the net, see [KJ87]. For instance, the number of tokens in each place of the TPNs of figure (2.1) is bounded by 1, which is the max number in its initial marking.

$$\forall k \in [\![0, \mathrm{K}]\!], \forall i \in [\![1, \mathrm{M}]\!],\ s_{m_{ki}} \in [\![0, M_{max}]\!] \tag{3.14}$$

These improvements are used to reduce search space and can also be seen as additional constraints to the CP model. Based on all these considerations above,

we will give the full constraint programming model.

3.2.3 Constraint Programming Model

Based on all expressions above, a full constraint programming model is given in figure (3.3). Equations (3.15 - 3.21) are used to initialize the initial and final states of TPNs. Meanwhile, equations (3.30 - 3.34) give the domains of all *variables* of this model. Finally, equations (3.22 - 3.26) represents the updating of marking vectors and residual duration vectors, which can be seen as constraints between all variables in a constraint programming approach view. Equations (3.27 - 3.29) represent improvements.

3.2.4 Numerical Experiments

To validate our approach and give a visible expression of following techniques, benchmarks for our basic $CP(\kappa)$ model are given. This basic CP model contains all posted constraints and improvements of section (3.2). Then the effects of all improvements will be given separately.

Experiments were carried out using a 2.93 Ghz Pentium with 4 Go of RAM, using the constraint programming tool *Ilog Solver*. We have used the Timed PNs presented in figure (3.2). The goal was to get the first feasible solution leading from s_0 to s_f. Note that there are 2 tokens in each beginning place of jobs.

In the following tables, we give the (*time*) to obtain the first solution, the number of inconsistent choices (*fails*), the (variables) (Abbreviation as Vars) and the (*constraints*) (Abbreviated as Cons) that participate in searching for a solution, the total time needed to reach the final marking (*makespan*) (Abbreviated as Mksp) and the (*first firing sequence found*) (Abbreviated as FirstS) found. If the same solution appears several times, we give it a name using prefix ϕ and make references to it later to save space.

3.2.4.1 Influence of the Search Depth

First we assess the influence of the number of firing dates κ used to build the incremental model. The results are given in table (3.1). The order of labeling variables are not given, and the *Defaut* manner is used. In fact, the default manner will be

Let (R, d) be a TPN with its initial state s_0 given at date v_0. Let s_f be a target state. The constraint programming model $CP(\mathsf{K})$ is defined by:

$$\forall k \in [\![0, \mathsf{K}-1]\!], \forall i \in [\![1, \mathrm{M}]\!], \forall j \in [\![1, \mathrm{N}]\!]$$

initial information

$$s_{m0i} = m_0(i) \tag{3.15}$$
$$s_{m\mathsf{K}i} = m_f(i) \tag{3.16}$$
$$s_{r0j} = r_0(j) \tag{3.17}$$
$$s_{r\mathsf{K}j} = r_f(j) \tag{3.18}$$
$$\sigma_{0j} = 0 \tag{3.19}$$
$$\sigma_{Kj} = 0 \tag{3.20}$$
$$T^f_{0j} = 0 \tag{3.21}$$

update residual duration vector

$$s_{rkj} - \Delta_{v_k} > 0 \Rightarrow s_{r(k+1)j} = s_{rkj} - \Delta_{v_k} \wedge \quad \sigma_{(k+1)j} = 0 \tag{3.22}$$
$$s_{rkj} - \Delta_{v_k} \leqslant 0 \Rightarrow s_{r(k+1)j} = d_j \cdot \sigma_{(k+1)j} \tag{3.23}$$

update marking

$$s_{rkj} > 0 \wedge \quad s_{rkj} - \Delta_{v_k} \leqslant 0 \Rightarrow T^f_{(k+1)j} = 1 \tag{3.24}$$
$$s_{rkj} \leqslant 0 \vee \quad s_{rkj} - \Delta_{v_k} > 0 \Rightarrow T^f_{(k+1)j} = 0 \tag{3.25}$$
$$s_{m(k+1)i} - s_{mki} = \sum_{c=0}^{\mathrm{N}-1} C^+_{ic} \cdot T^f_{(k+1)c} - \sum_{c=0}^{\mathrm{N}-1} C^-_{ic} \cdot \sigma_{(k+1)c} \tag{3.26}$$
$$\forall k \in [\![0, \mathsf{K}]\!], \sum_{j=0}^{\mathrm{N}-1} \sigma_{(k+1)j} \neq 0 \tag{3.27}$$

modeling improvements and initial domains

$$\Delta_{v_k} \leqslant \max_{j \in [\![1,\mathrm{N}]\!]} s_{rkj} \tag{3.28}$$
$$s_{r(k+1)j} \leqslant d(j) \tag{3.29}$$
$$\sigma_{(k+1)j} \in \{0, 1\} \tag{3.30}$$
$$T^f_{(k+1)j} \in \{0, 1\} \tag{3.31}$$
$$s_{r(k+1)j} \in [\![0, D_{max}]\!] \tag{3.32}$$
$$s_{m(k+1)i} \in [\![0, M_{max}]\!] \tag{3.33}$$
$$\Delta_{v_k} \in]\!]0, D_{max}]\!] \cup \Delta_{v_0} \in [\![0, D_{max}]\!] \tag{3.34}$$

Figure 3.3: Constraint Programming Model

the order they occur in the sequence $\sigma_1 \Delta_{v_0} s_{m_1} s_{r_1} \sigma_2 \Delta_{v_1} s_{m_2} s_{r_2} \ldots \sigma_{\mathrm{K}} \Delta_{v_{\mathrm{K}-1}} s_{m_{\mathrm{K}}} s_{r_{\mathrm{K}}}$ since equations are executed step by step recursively in the incremental approach.

Since each beginning place of jobs has two tokens, each transition must be

fired two times to lead from s_0 to s_f. Thus, any solution should contain no more than 20 transition firings *plus* the final empty step (cf. notes for eq. (3.12)). This consideration explains why no result is found if $\mathrm{K} > \mathrm{K}_{max} = 21$. Note that even if simple considerations lead to this result, the constraint programming resolution does not finish after a long time, since the search tree must be traversed entirely to decide there is no solution.

The first number of steps for which a sequence exists is $\mathrm{K}_{min} = 10$. The time given for $\mathrm{K} = 9$ corresponds to the duration to traverse the whole search tree without finding a solution.

Starting from $\mathrm{K}_{min} = 10$ until K_{max}, the first solution is found earlier when the number of steps is greater. Obviously, changing the number of steps from K to $\mathrm{K}+1$ does not add nor change constraints over the variables belonging to the first K steps, thus the solutions are tried out in the same order in the respective search trees, since the labeling order follow the order of steps. Moreover, the branching mechanism used to build the search tree starts by labeling *one transition by step* (the minimum possible number of transitions firing, from equation (3.12). When the maximal number of steps has been reached, since $\mathrm{K} < \mathrm{K}_{max}$, the final marking has not been reached yet, thus the constraint programming algorithm backtracks and continues the search. When sequences containing one transition firing at a time have all been considered, the search mechanism labels sequences containing two transitions for the last step, and so forth. Finally, the less steps we have to build a solution, the greater number of backtracks is needed to find a solution where sufficient transition firings have been merged into steps, and the longer is the search procedure.

In table (3.1), for a default labeling strategy is applied, variables that appeared earlier will be closer to the root node. Meanwhile, all labeling strategies choose values in the variable's domain in ascending order. Therefore, Solver goes down in the search tree and backtrack to variables that appeared late. That is the reason why all firing sequences begin with transition o_{31}.

The previous results were obtained using default labeling strategy, corresponding to label variables in the order of they occur. In the next section, we assess the efficiency of other generic labeling strategies.

K	Time	Fails	Vars	Cons	Mksp	FirstS
9	33.35 s	830245	477	no solution exists		
10	10.51 s	273295	525	886	30	o_{31}, $o_{31}o_{32}$, $o_{21}o_{32}$, $o_{11}o_{21}o_{22}$, $o_{22}o_{23}$, $o_{23}o_{24}$, $o_{11}o_{12}o_{24}$, $o_{12}o_{13}o_{33}$, $o_{13}o_{33}$
11	1.76 s	41936	573	971	32	o_{31}, o_{32}, o_{31}, $o_{21}o_{32}$, $o_{11}o_{21}o_{22}$, $o_{22}o_{23}$, $o_{23}o_{24}$, $o_{11}o_{12}o_{24}$, $o_{12}o_{13}o_{33}$, $o_{13}o_{33}$
12	0.42 s	9907	621	1058	34	$\phi_1 = o_{31}$, o_{32}, o_{33}, o_{31}, $o_{21}o_{32}$, $o_{11}o_{21}o_{22}$, $o_{22}o_{23}$, $o_{23}o_{24}$, $o_{11}o_{12}o_{24}$, $o_{12}o_{13}$, $o_{13}o_{33}$
13	0.14 s	2829	669	1146	37	o_{31}, o_{32}, o_{33}, o_{31}, o_{32}, o_{21}, $o_{11}o_{21}o_{22}$, $o_{22}o_{23}$, $o_{23}o_{24}$, $o_{11}o_{12}o_{24}$, $o_{12}o_{13}$, $o_{13}o_{33}$
14	0.14 s	1640	717	1233	39	o_{31}, o_{32}, o_{33}, o_{31}, o_{32}, o_{33}, o_{21}, $o_{11}o_{21}o_{22}$, $o_{22}o_{23}$, $o_{23}o_{24}$, $o_{11}o_{12}o_{24}$, $o_{12}o_{13}$, o_{13}
15	0.06 s	115	765	1315	41	o_{31}, o_{32}, o_{33}, o_{31}, o_{32}, o_{33}, o_{21}, o_{22}, o_{23}, $o_{11}o_{21}o_{24}$, $o_{11}o_{12}o_{22}$, $o_{13}o_{23}$, $o_{12}o_{24}$, o_{13}
16	0.01 s	30	813	1403	40	o_{31}, o_{32}, o_{33}, o_{31}, o_{32}, o_{33}, o_{21}, o_{22}, o_{23}, o_{24}, $o_{11}o_{21}$, $o_{11}o_{12}o_{22}$, $o_{13}o_{23}$, $o_{12}o_{24}$, o_{13}
17	0.04 s	23	861	1491	46	o_{31}, o_{32}, o_{33}, o_{31}, o_{32}, o_{33}, o_{21}, o_{22}, o_{23}, o_{24}, o_{21}, o_{11}, $o_{11}o_{12}o_{22}$, $o_{13}o_{23}$, $o_{12}o_{24}$, o_{13}
18	0.02 s	0	909	1580	48	o_{31}, o_{32}, o_{33}, o_{31}, o_{32}, o_{33}, o_{21}, o_{22}, o_{23}, o_{24}, o_{21}, o_{22}, o_{23}, o_{11}, $o_{11}o_{12}o_{24}$, $o_{12}o_{13}$, o_{13}
19	0.05 s	0	957	1668	51	o_{31}, o_{32}, o_{33}, o_{31}, o_{32}, o_{33}, o_{21}, o_{22}, o_{23}, o_{24}, o_{21}, o_{22}, o_{23}, o_{24}, o_{11}, $o_{11}o_{12}$, $o_{12}o_{13}$, o_{13}
20	0.03 s	1	1005	1753	53	o_{31}, o_{32}, o_{33}, o_{31}, o_{32}, o_{33}, o_{21}, o_{22}, o_{23}, o_{24}, o_{21}, o_{22}, o_{23}, o_{24}, o_{11}, o_{12}, o_{11}, $o_{12}o_{13}$, o_{13}
21	0.03 s	0	1053	1839	56	o_{31}, o_{32}, o_{33}, o_{31}, o_{32}, o_{33}, o_{21}, o_{22}, o_{23}, o_{24}, o_{21}, o_{22}, o_{23}, o_{24}, o_{11}, o_{12}, o_{13}, o_{11}, o_{12}, o_{13}
22	>6000 s	no solution exists				

Table 3.1: Changing the Number of Firing Dates

3.2.4.2 Influence of Improvements

In order to compare the effect of different situations, we have chosen to consider executions of $K = 12$ firing dates containing the final empty timed steps to elapse the time for finish the firing transitions. To show the effect of all the improvements, the results are given in table (3.2). If all improvements are removed, the solver will run out of memory. Therefore we reduce one improvement each time from the basic CP model to show their contributions of reducing the search space.

In the table (3.2), the *time* consumed and the *fails* encountered will mainly represent the efficiency. The first line presents the result with keeping all improvements, and it finds the solution the fastest. It means that every improvement can reduce the search space to enhance the efficiency.

Lines 2, 3, 4 represent the effect of making improvements on residual duration vector $\overrightarrow{s_{r_k}}$. Equation (3.8) is used to define the domains of residual duration vector, which must be smaller than the max duration. Equation (3.9) continue

	Improvements	Time	Fails	Vars	Cons	Mksp	FirstS
1	All	0.42 s	9907	621	1058	34	ϕ_1
2	All-equation (3.8)	0.55 s	10595	621	1057	34	ϕ_1
3	All-equation (3.9)	0.55 s	10817	621	1057	34	ϕ_1
4	All-equations (3.8, 3.9)	0.70 s	15338	621	1057	34	ϕ_1
5	All-equation (3.10)	0.53 s	10595	621	1058	34	ϕ_1
6	All-equation (3.11)	3.96 s	99896	621	1046	34	ϕ_1
7	All-equation (3.13)	0.86 s	15739	621	1058	34	ϕ_1
8	All-equations (3.10, 3.11, 3.13)	not enough memory					
9	Equations (3.10, 3.11, 3.13)	13.04 s	276163	610	1053	30	ϕ_2
10	All-equation (3.12)	8.96 s	200468	610	1054	34	ϕ_2
11	All-equation (3.14)	0.55 s	10595	621	1058	34	ϕ_1
12	No improvements	not enough memory					

$\phi_2 = o_{31}, \varnothing, o_{32}, o_{31}, o_{21}o_{32}, o_{11}o_{21}o_{22}, o_{22}o_{23}, o_{23}o_{24}, o_{11}o_{12}o_{24}, o_{12}o_{13}o_{33}, o_{13}o_{33}$

Table 3.2: Influence of Improvements

to precise the domain of each transition's residual duration variable, which can be seen as constraints. Since $D_{max} = max(\overrightarrow{d})$, equation (3.9) contain the effect of equation (3.8). Therefore line 3 meets more fails than line 2. But the residual duration vector is not taking the most important role. Improvements of residual duration vector do not enhance too much efficiency, which means residual duration vector does not influence the solution too much.

Lines 5, 6, 7, 8, 9 represent the effect of making improvements on time elapse between two firing dates $\Delta_{v_{k-1}}$. Equation (3.10) defines the domain of $\Delta_{v_{k-1}}$ and equation (3.11) restrict $\Delta_{v_{k-1}}$ smaller than the longest residual duration at date v_{k-1}. These two equations implies that all fired transitions will be finished at date $v_{k-1} + Max(\overrightarrow{s_{r_{k-1}}})$. Therefore all tokens needed to fire σ_k are available, so $\Delta_{v_{k-1}}$ do not need to make more time elapse. Line 6 meets more fails, for $\Delta_{v_{k-1}}$ can lead solver go deep in the search tree with bigger domains. If solver cannot find one solution at some open node of search tree, increasing $\Delta_{v_{k-1}}$ cannot change the situation, but just make solver meet more fails. Equation (3.13) represent a timed firing step cannot be fired with no time elapse, in order to forbid two steps to be merged into one step under this situation. So line 7 meets a little more fails with line 1 for the continuous steps. Meanwhile Equation (3.13) just affect the number of steps, so it does not bring too much efficiency. Line 8 presents that solver will run out of memory with no $\Delta_{v_{k-1}}$ improvement and line 9 presents

that only with improvements of $\Delta_{v_{k-1}}$ can lead solver from *run out of memory* to one solution. These mean that the improvements for reducing the domains of $\Delta_{v_{k-1}}$ has taken the most important role for reducing search space. Meanwhile a bad strategy with labeling $\Delta_{v_{k-1}}$ will lead solver go deep in the search tree and meet explosive fails.

Line 10 represents the effect of making improvements on the firing transitions σ_k at each step. Equation (3.12) means forbid the empty steps. If empty step allowed, solver can make empty steps at the top of search tree and will compress transitions at the following steps. As shown in subsection (3.2.4.1), less steps will lead solver to meet more fails. Meanwhile, empty steps at the top of search tree will make solver go deeper and backtrack more nodes. So solver meets nearly 20 times fails than line 1.

At last, line 11 represents the effect of making improvements on marking. As said about *p-semi flows*, tokens in each place will be bounded with the number of initial marking. So this improvement does not make much efficiency.

To conclude, all improvements have reduced search space, and implied that $\Delta_{v_{k-1}}$ will take important role in finding a solution. For now, we cannot continue to improve the efficiency of finding one feasible solution through making improvements. Therefore we begin to introduce the search strategies, which will lead the search in a controlled way and give more efficiency.

3.3 Search Strategies

After modeling the incremental model with CP, the reachability problem is needed to be solved using the method defined in definition (2.15). If there exists a solution, we can find one K to find it as described in chapter (2). But the *time* needed to find it varies greatly with respect to *search space* and *search strategies*. Here search strategies means all the methods that can improve the search efficiency. In our paper [HBYT12c], we have shown many benchmarks and discuss different search strategies as in this thesis.

To express all the search strategies and to show their efficiency, we follow the figure (3.2) to illustrate.

3.3.1 Variables Ordering

The efficiency of a CP approach is mainly based on the labeling strategies used to choose which variables are enumerated first. In Solver, there are several *generic strategies*. Generic strategies are based on the size of the domain of variables. One can cite:

FirstUnbound Label variables in the order they occur;

MinSize Label the variable with the smallest domain first;

MaxSize Label the variable with the largest domain first;

MinMax Label the variable with the least maximal bound first;

MinMin Label the variable with the least minimal bound first;

MaxMin Label the variable with the greatest minimal bound first;

MaxMax Label the variable with the greatest maximal bound first;

MinRegretMin Start by labeling the variable for which the difference between the smallest possible value and the next smallest value is minimum, etc.

For full list, reader can follow [IBM10].

But generic strategies do not consider the structure information of TPNs. This cannot give the best search strategies and let us to formulate several controlled search strategies. So in this thesis, we propose also to develop *dedicated labeling strategies* based on our knowledge of the TPN behavior and the meaning of variables. In our model, decision variables belong mainly to 4 classes:

- Step firing variables ($\overrightarrow{\sigma_k}$)
- Firing Dates Variables (Δ_{v_k})
- Intermediate Marking Variables ($\overrightarrow{s_{m_k}}$)
- Intermediate Residual Duration Variables ($\overrightarrow{s_{r_k}}$)

One must note that there exist different kinds of *dependency relations* between these variables. Indeed, if one choose values for (σ_k), some variables (s_{m_k}) and (s_{r_k}) are affected to traduce the consumption of tokens, but the production of tokens still depends on the firing dates, thus variables are simply more constrained. Enumerating variables (Δ_{v_k}) does not imply any subsequent instantiation of other variables. On the contrary, enumerating marking have an effect on all other variables since it implies dates for the production of tokens.

All labeling strategies choose values in the domain of variable by ascending order. This strategy is default in Ilog Solver.

3.3.1.1 Generic Labeling Strategies

In order to compare the effect of different generic search strategies, we have chosen to consider executions containing $\kappa = 12$ firing dates. All variables of σ_k, Δ_{k-1}, s_{m_k}, s_{r_k} are added into one vector $AllV$ instead of label each kind of vectors one by one. Thus the generic labeling strategies can affect them with no tendency.

As the generic labeling strategies are mainly based on choosing different domains, the domains of labeled variables are given in equations (3.30, 3.32, 3.33 and 3.34).

	Search Strategy	Time	Fails	Vars	Cons	Mksp	FirstS
1	FirstUnbound	0.51 s	10595	621	1058	34	ϕ_1
2	MinSize	0.49 s	10085	621	1058	34	ϕ_1
3	MinMax	0.47 s	9356	621	1058	34	ϕ_1
4	MinMin	0.42 s	8800	621	1058	34	ϕ_1
5	MaxSize	15586.7 s	586386643	621	1061	35	ϕ_3
6	MaxMin	6048.55 s	218786627	621	1058	29	ϕ_4
7	MaxMax	1201.69 s	62351483	621	1063	34	ϕ_5
8	MinRegretMin	0.53 s	10595	621	1058	34	ϕ_1

$\phi_3 = o_{31}, o_{31}, o_{21}, o_{21}o_{22}, o_{22}o_{23}, o_{23}o_{24}, o_{24}, o_{11}o_{32}, o_{11}o_{12}o_{32}, o_{12}o_{13}o_{33}, o_{13}o_{33}$
$\phi_4 = o_{11}o_{21}, o_{31}, o_{22}, o_{21}, o_{23}o_{32}, o_{22}, o_{23}o_{24}, o_{24}o_{31}, o_{11}o_{12}o_{32}, o_{12}o_{13}o_{33}, o_{13}o_{33}$
$\phi_5 = o_{11}, o_{31}, o_{21}, o_{21}o_{22}, o_{22}o_{23}, o_{23}o_{24}, o_{24}, o_{31}o_{32}, o_{11}o_{12}o_{32}, o_{12}o_{13}o_{33}, o_{13}o_{33}$

Table 3.3: Generic Labeling Strategies

The results given in table (3.3) can be mostly explained by the labeling order between variables $(\Delta_{v_{k-1}})$ and (σ_k), since $(\Delta_{v_{k-1}})$ and (σ_k) will give the most influence with these results as shown in table (3.2). Indeed: if the chosen labeling

strategy leads to enumerate values for variables $(\Delta_{v_{k-1}})$ first, since the enumeration is made in ascending order, the exploration process will start by using small values for variables $(\Delta_{v_{k-1}})$, searching for solutions with increasing values of $v_{\text{K}} = \sum_{k=0}^{\text{K}-1} \Delta_{v_k}$ starting from value K (since equation (3.13) forbids $\Delta_{v_k, k \geqslant 1} = 0$). Such exploration implies that the first solution found has the smallest makespan v_{min}. However, finding this solution requires a long time since many situations must be explored first with $v_{\text{K}} < v_{min}$. Since choosing values for variables $(\Delta_{v_{k-1}})$ does not constrain much variables σ_k, large subtrees need to be explored before the constraint programming solver can conclude choices for $(\Delta_{v_{k-1}})$ were bad, leading to many fails.

The situation described above occurs when using *"MaxSize"*, *"MaxMin"* and *"MaxMax"* strategies. Indeed: domains of variables (σ_k) are restricted to $\{0, 1\}$, while domains for variables $(\Delta_{v_{k-1}})$ belong generally to $[\![1, D_{max}]\!]$. We have noticed that after a few propagation of contraints, domains of variables $(\Delta_{v_{k-1}})$ keep being wider and with greater bounds than domains of variables (σ_k), leading to explore the search tree by enumerating variables $(\Delta_{v_{k-1}})$ firstly using the considered labelling strategies.

On the contrary, other labelling strategies imply a better mix between variables $(\Delta_{v_{k-1}})$ and (σ_k), leading to obtain the same solution with different explorations. Times and number of fails obtained may vary by a factor of 10.

In order to get a better understanding of these explorations, we propose to evaluate dedicated labeling strategies based on a fine ordering of decision variables $(\sigma_k, \Delta_{v_{k-1}}, s_{m_k}, s_{r_k})$. Some preliminary observations have already been made in section (3.3.1) to point out the relations of these variables one with each other and the effects an affectation of one variable can have on several others.

In the following sections, we verify experimentally these considerations using several labeling orders, *"globally"* for the whole set of variables of one class, or *"one step at a time"*. For example, labeling $\sigma + \Delta$ globally means considering the labelling order: $\sigma_1 \sigma_2 \dots \sigma_{\text{K}} \Delta_{v_0} \Delta_{v_1} \dots \Delta_{v_{\text{K}-1}}$. Labeling $\sigma + \Delta$ *"step-by-step"* means considering the order $\sigma_1 \Delta_{v_0} \sigma_2 \Delta_{v_1} \dots \sigma_{\text{K}} \Delta_{v_{\text{K}-1}}$. To distinguish with them, asterisk is added as $(\sigma + \Delta)^*$ to represent labeling $\sigma + \Delta$ globally.

3.3.1.2 Dedicated Labeling Strategies - Global Labeling

A study of possible global dedicated strategies is given in table (3.4). Observations given in previous sections are still valid: when variables (Δ_{k-1}) are enumerated first, inconsistencies appear lately while traversing the search tree, since many choices can be made before a constraint is broken.

Interesting facts appear when enumerating variables (s_{m_k}) or (s_{r_k}) first. One should note that both classes of variables share many constraints with variables (σ_k) and $(\Delta_{v_{k-1}})$. However, the implications of affecting variables (s_{r_k}) or (s_{m_k}) are not the same, as can be seen in equations (2.9) and (2.10).

First, variables (s_{r_k}) have such strong links with other variables that labeling two successive vectors (s_{r_k}) and $(s_{r_{k+1}})$ implies values for Δ_{v_k}, σ_{k+1} and therefore $(s_{m_{k+1}})$. Labeling first all (s_{r_k}) reduce quickly the domain of other variables towards the value they should finally have, which make strategies starting by labeling (s_{r_k}) the most effective ones within global labeling.

To the contrary, labelling variables (s_{m_k}) does not have a big impact on reducing the domain of other variables, since knowing intermediate markings does not imply knowing the time when they are reached nor the exact firing sequence used. Thus, after having labeled the set of intermediate markings (s_{m_k}), the constraint solver must search for values of variables $(\Delta_{v_{k-1}})$, (σ_k) to finally find no solution exists, which implies a deeper tree traversal and prohibitive delays.

	Search Strategy	Time	Fails	Vars	Cons	Mksp	FirstS
1	$(\sigma + \Delta)^*$	0.56 s	22805	621	1058	34	ϕ_1
2	$(\sigma + s_m + \Delta)^*$	0.59 s	21836	621	1058	34	ϕ_1
3	$(\sigma + s_r + \Delta)^*$	0.67 s	21836	621	1058	34	ϕ_1
4	$(\Delta + \sigma)^*$	476.54 s	14838186	621	1066	28	ϕ_6
5	$(\Delta + s_m)^*$	488.81 s	14838660	621	1066	28	ϕ_6
6	$(s_m + \sigma + \Delta)^*$	no solution found after 6000s					
7	$(s_m + \Delta)^*$	no solution found after 6000s					
8	$(s_r + \Delta)^*$	0.42 s	9907	621	1508	34	ϕ_1
9	$(s_r + \sigma + \Delta)^*$	0.42 s	9907	621	1508	34	ϕ_1
10	$(s_r + s_m + \Delta)^*$	0.42 s	9907	621	1508	34	ϕ_1

$\phi_6 = o_{31}$, o_{21}, o_{32}, o_{31}, o_{22}, $o_{21}o_{23}o_{32}$, $o_{11}o_{22}$, $o_{23}o_{24}$, $o_{11}o_{12}o_{24}$, $o_{12}o_{13}o_{33}$, $o_{13}o_{33}$

Table 3.4: Dedicated Global Labeling Strategies

3.3.1.3 Dedicated Labeling Strategies - Step by Step Labeling

Results corresponding to labeling strategies following the order of firing sequences are given in table (3.5). They lead generally to the best results, since inconsistencies are detected sooner and the considered TPN does not contain *deadlocks*. Thus, any fireable step labeled allows the system to get closer to the final marking and can be used to produce the final solution without needs for backtracking. This is particularly true when considering strategies starting by labeling (s_{m_k}), which find a solution using very few backtracks.

	Search Strategy	Time	Fails	Vars	Cons	Mksp	FirstS
1	$\sigma + \Delta$	0.51 s	10595	621	1058	34	ϕ_1
2	$\sigma + s_m + \Delta$	0.53 s	10572	621	1058	34	ϕ_1
3	$\sigma + s_r + \Delta$	0.55 s	11066	621	1058	34	ϕ_1
4	$\Delta + \sigma$	2.54 s	67532	621	1067	30	ϕ_7
5	$\Delta + s_m$	0.09 s	1762	621	1067	30	ϕ_8
6	$s_m + \Delta$	0.03 s	6	621	1064	32	ϕ_9
7	$s_m + s_r$	0.06 s	10	621	1064	30	ϕ_9
8	$s_r + \Delta$	0.39 s	9758	621	1508	34	ϕ_1
9	$s_r + \sigma + \Delta$	0.39 s	9820	621	1508	34	ϕ_1
10	$s_r + s_m + \Delta$	0.42 s	9771	621	1508	34	ϕ_1

$\phi_7 = o_{31}, o_{21}, o_{32}, o_{31}, o_{32}, o_{11}o_{21}o_{22}, o_{22}o_{23}, o_{23}o_{24}, o_{11}o_{12}o_{24}, o_{12}o_{13}o_{33}, o_{13}o_{33}$
$\phi_8 = o_{11}o_{21}, o_{11}, o_{12}o_{22}, o_{31}\ o_{12}o_{13}o_{23}, o_{13}o_{21}o_{31}, o_{22}, o_{23}o_{24}, o_{32}, o_{32}o_{33}, o_{24}o_{33}$
$\phi_9 = o_{11}o_{21}, o_{11}o_{12}o_{22}, o_{12}o_{13}o_{23}, o_{13}o_{21}o_{31}, o_{31}, o_{22}, o_{23}o_{24}, o_{24}, o_{32}, o_{32}o_{33}, o_{33}$

Table 3.5: Dedicated Step-by-Step Labeling Strategies

3.3.2 Values Ordering

After choosing a variable as a open node in the search tree, values of this variable are enumerated one by one. In general, enumerating values will be executed from low to high. But in our model, we would like to enumerate from high to low, since this strategy will be more efficient and take more physical sense. For example, $\sigma_{kj} \in [0, 1]$, if $\sigma_{kj} = 1$, it means this transition t_j will be fired at date v_k, otherwise $\sigma_{kj} = 0$ will waste much time on useless empty firing. For example the solution ϕ_1, o_{31} is first fired, and timed steps firings will be compressed at the last part of steps.

In the physical sense, a firing sequence is the aim to be got. and we would like to enumerate values of σ_k and s_{mk} from high to low. In fact, if a solution is found at a lower Δ_{k-1}, a new solution can be found by adding time elapse to Δ_{k-1}.

To illustrate the efficiency of using the "high to low" value ordering, we choose several representative benchmarks in the previous section as shown in table (3.6). All variables are using "step-by-step" labeling strategies except the labeling strategies with asterisk, which are using global labeling strategies. For all experiments K = 12, except for several pointed out strategies.

In table (3.6), if σ is labeled first, a solution can be found easily, since there is no *loops* or *deadlocks* in the structure of TPN, Solver will find one solution with no more than 1 fails. But if Δ is labeled first, finding one solution can also meet many fails. Especially if label σ from high to low, Solver will meet more fails, since $\sigma_{kj} = 1$ will lead Solver to the deep deadend, however $\sigma_{kj} = 0$ will meet deadend earlier and backtrack to a solution.

To conclude, a good strategy will be dedicated step-by-step labeling strategy with labeling σ_k from high to low, labeling s_{m_k} from high to low and s_{r_k} from low to high, and labeling Δ_{k-1} from low to high to generate the firing dates of σ_k. In fact, $\sigma_k + \Delta_{k-1}$ with dedicated step-by-step labeling strategies can generate all information in firing sequences. Here labeling s_{m_k} and s_{r_k} is to verify the intermediate state in order to find fails earlier and backtrack.

Note that in [DYKG04], author also finds the first firing sequence by finding all possible untime firing sequences, then restricting these untime firing sequences with time propagation. This method is similar to results line 9 in table (3.6).

3.3.3 Backtracking

In practice, the main backtracking search algorithms are derived from [IBM10] as follow.

- Depth-First Search (DFS) DFS is the standard search procedure. Depth-First Search explores the search tree in a "left-to-right" fashion. It searches by selecting the left leaf at each open node until the end of the search space is reached. If it does not find a solution, it backtracks and tries again.

	Search Strategy	Time	Fails	Vars	Cons	Mksp	FirstS
1	$\sigma + \Delta$ K = 9	133.1 s	7505863	477	no solution exists		
2	$\sigma + \Delta$ K = 10	0.06 s	0	516	875	30	ϕ_{12}
3	$\sigma + \Delta$-(Impros Δ)	0.05 s	1	621	1042	34	ϕ_{10}
4	$s_m + \Delta$-(Impros Δ)	5.23 s	146006	621	1046	34	ϕ_1
5	$\sigma + \Delta$-(Impros σ)	0.05 s	0	610	1061	1051	ϕ_{11}
6	$\sigma + \Delta$	0.05 s	1	621	1062	32	ϕ_{10}
7	$s_m + \Delta$	0.45 s	10621	621	1058	34	ϕ_1
8	$\Delta + \sigma$	0.09 s	1762	621	1067	30	ϕ_8
9	$\sigma + \Delta^*$	0.02 s	1	621	1062	32	ϕ_{10}
10	$\Delta + \sigma^*$	1173.17 s	42860426	621	1068	28	ϕ_{13}

$\phi_{10} = o_{11}o_{21}$, $o_{11}o_{12}o_{22}$, $o_{12}o_{13}o_{23}$, $o_{13}o_{21}o_{31}$, $o_{22}o_{31}$, $o_{23}o_{24}$, o_{24}, o_{32}, o_{32}, o_{33}, o_{33}
$\phi_{11} = o_{11}o_{21}$, $o_{11}o_{12}o_{22}$, $o_{12}o_{13}o_{23}$, $o_{13}o_{21}o_{31}$, $o_{22}o_{31}$, $o_{23}o_{24}$, o_{24}, o_{32}, $o_{32}o_{33}$, o_{33}
$\phi_{12} = o_{11}o_{21}$, $o_{11}o_{12}o_{22}$, $o_{12}o_{13}o_{23}$, $o_{13}o_{21}o_{31}$, $o_{22}o_{31}$, $o_{23}o_{24}$, o_{32}, $o_{32}o_{33}$, $o_{24}o_{33}$
$\phi_{13} = o_{31}$, o_{21}, o_{32}, o_{11}, $o_{21}o_{22}o_{31}$, $o_{23}o_{32}$, $o_{11}o_{12}o_{22}$, $o_{23}o_{24}$, $o_{12}o_{24}o_{33}$, $o_{13}o_{33}$, o_{13}

Table 3.6: Comparing Generic Labeling Strategies

- Best First Search (BFS) The Solver implementation uses a parameter ε. When selecting an open node, Solver determines the set of open nodes the cost of which is at most $(1 + \varepsilon)$ worse than the best open node. If the child of the current node is in the set, Solver goes to this child.

- Slice-Based Search (SBS) The *discrepancy* of a search node is defined as the number of right moves in the search tree (as shown in figure (3.1). Given a parameter step, slice-based search will first explore nodes with a discrepancy less than step. After this exploration is complete, it will explore nodes with a discrepancy between step and 2*step, and so on. This search strategy cuts the search tree into slices.

- Depth-Bounded Discrepancy Search (DDS) DDS makes the assumption that mistakes are made more likely near the top of the search tree than further down. For this reason, this procedure does not count the number of discrepancies but the depth of the last one. Given a parameter step, DDS will first explore nodes with a depth less than step. After this exploration is complete, it will explore nodes with a depth between step and 2*step, and so on.

To show how these backtrack strategies work, results are given in table (3.7) with K = 12. All variables are labeled with value ordering from low to high to make more fails happen in order to see the effect of backtracking strategies.

In lines 1 and 2, the fails and choice points are nearly the same, because Solver will always propagate all nodes that met with DFS, then backtrack. However SBS and DDS will backtrack based on the depth of discrepancy or nodes depth. These strategies will not branch all open nodes, which lead Solver meet more choice points than fails. In lines 3 and 4, Solver will first branch open nodes with lowest right move (one right move is seen as a discrepancy). This strategy will be more proper for right values situated in the last part of one variable. In line 5 and 6, Solver will branch open nodes less than n*Step depth. This strategy can avoid Solver going to deep deadends with no solution. But both SBS and DDS are not too proper to cooperate with dedicate labeling strategies in our model.

	Search Strategy	Time	Fails	Choice Points	Vars	Mksp	FirstS
1	$(\sigma + \Delta)$ DFS	0.51 s	10595	10611	621	34	ϕ_1
2	$(\sigma + \Delta)^*$ DFS	0.56 s	22805	22821	621	34	ϕ_1
3	$(\sigma + \Delta)$ SBS step = 4	7.92 s	62172	368699	621	34	ϕ_{14}
4	$(\sigma + \Delta)^*$ SBS step = 4	6.38 s	65335	413006	621	34	ϕ_{14}
5	$(\sigma + \Delta)$ DDS step = 4	0.89 s	4687	40329	621	32	ϕ_{15}
6	$(\sigma + \Delta)^*$ DDS step = 4	0.87 s	6145	53860	621	36	ϕ_{16}

$\phi_{14} = o_{11}o_{21}$, $o_{21}o_{22}o_{31}$, o_{32} , o_{33}, o_{31} , o_{32}, $o_{22}o_{23}$, $o_{23}o_{24}$, $o_{11}o_{12}o_{24}$, $o_{12}o_{13}$, $o_{13}o_{33}$
$\phi_{15} = o_{11}o_{21}$, $o_{11}o_{12}o_{22}$, $o_{12}o_{13}o_{23}$, o_{31} , o_{32}, o_{33}, o_{24}, $o_{13}o_{21}$, $o_{22}o_{31}$, $o_{23}o_{32}$, $o_{24}o_{33}$
$\phi_{16} = o_{11}o_{31}$, $o_{11}o_{12}$, $o_{12}o_{13}o_{31}$, o_{32}, o_{33}, $o_{13}o_{31}$, o_{32}, $o_{21}o_{22}$, $o_{22}o_{23}$, $o_{23}o_{24}$, $o_{24}o_{33}$

Table 3.7: Backtracking Strategies

Finally, we have developed several search strategies in our model. The most important one will be labeling variables using *step-by-step* manner, which is more closer to the physical sense of TPNs. Meanwhile, labeling variables (σ_k and s_{mk} with value ordering from high to low and Δ_k and s_{rk} with value ordering from low to high) will improve much the efficiency.

3.4 Linearization of Firing Sets Expressions

Since variables in conditional constraints will be only labeled after other variables are instantiated, we want to introduce a linearization technique to model the non-

linear equations (2.17 ,2.18). We follow the method of [BH06] to use the following conventions. If $\overrightarrow{x}$ is a vector from $\mathbb{Z}^k$, they denote by:

- $\overrightarrow{x}^+ \in \mathbb{N}^k$ the vector of its positive components, such that $\overrightarrow{x}^+(i) = \overrightarrow{x}(i)$ if $\overrightarrow{x}(i) > 0$ and 0 otherwise;
- $\overrightarrow{x}^s \in \mathbb{B}^k$ the vector representing its *sign*, such that: $\forall c \in [\![1, k]\!]$, $\overrightarrow{x}^s(c) = 0$ if $\overrightarrow{x}(c) \leqslant 0$ and $\overrightarrow{x}^s(c) = 1$ otherwise.

With linearization, the firing sets can be expressed by *positive* and *sign* conventions as follow:

$$T^f_{k+1} = \left\{ j \in [\![1, \mathrm{N}]\!], t_j \in \mathbb{T}, \left(s_{r_{kj}}\right)^s - \left(s_{r_{kj}} - \Delta_{v_k}\right)^s = 1 \right\} \tag{3.35}$$

$$T^s_{k+1} = \left\{ j \in [\![1, \mathrm{N}]\!], t_j \in \mathbb{T}, \left(s_{r_{kj}} - \Delta_{v_k}\right)^+ > 0 \right\} \tag{3.36}$$

Therefore, residual duration vector equation (2.10) and marking vector equation (2.9) are expressed as:

$$\overrightarrow{s_{r_{k+1}}} = \left(\overrightarrow{s_{r_k}} - \Delta_{v_k} \cdot \overrightarrow{I_d}\right)^+ + \sum_{t \in \mathbb{T}} d(t) \cdot \sigma_{k+1}(t) \cdot \overrightarrow{e_t} \tag{3.37}$$

$$\overrightarrow{s_{m_{k+1}}} = \overrightarrow{s_{m_k}} - C^- \cdot \overrightarrow{\sigma_{k+1}} + C^+ \cdot \left(\overrightarrow{s_{r_k}}^s - \left(\overrightarrow{s_{r_k}} - \Delta_{v_k} \cdot \overrightarrow{I_d}\right)^s\right) \tag{3.38}$$

Obviously, same equations remain valid if no transition is fired at the date v_{k+1} (i.e. $\overrightarrow{\sigma_{k+1}} = \overrightarrow{0}$), and allow to evaluate the *instantaneous state* at this date.

The *physical sense* of the equations is explained below:

- The quantity $\sum_{t \in \mathbb{T}} d(t) \cdot \sigma_{k+1}(t) \cdot \overrightarrow{e_t}$ represents the *new residual durations* coming from the execution of the step σ_{k+1} at the date $v + \Delta_v$;
- The quantity $\left(\overrightarrow{s_{r_k}} - \Delta_{v_k} \cdot \overrightarrow{I_d}\right)^+$ represents the *update* of the residual durations vector at the date v_{k+1}, from its value $\overrightarrow{s_{r_k}}$ at the date v_k. The $''+''$ operator allows to take into account only positive values;

- The quantity $C^- \cdot \overrightarrow{\sigma_{k+1}}$ represents the number of tokens removed from places upstream to the transitions of the step σ_{k+1}, the execution of which starts at the date v_{k+1} ;
- Finally, the quantity $C^+ \cdot \left(\overrightarrow{s_{r_k}}^s - \left(\overrightarrow{s_{r_k}} - \Delta_{v_k} \cdot \overrightarrow{I_d} \right)^s \right)$ represents the Parikh vector of the transitions, the firing of which ends at the date v_{k+1}. This expression is made from the comparison between the Parikh vector of the transitions that were pending at the date v_k (vector $\overrightarrow{s_{r_k}}^s$), and the Parikh vector of the transitions that *will be* active at the date v_{k+1} (vector $\left(\overrightarrow{s_{r_k}} - \Delta_{v_k} \cdot \overrightarrow{I_d} \right)^s$).

Now, the problem is to express these operators *positive* "+" and *sign* "s".

3.4.1 Formulation of "+" and "s" Operators

The operators *"+"* and *"s"* introduced in the previous section acts on *vectors objects*. However, they acts uniformly on each component of the input vector, and can be defined in a natural way using the corresponding *scalar* operators. [BH06] propose to address the calculation of values X^s and X^+ associated to an integer $X \in \mathbb{Z}$, such that:

$$X > 0 \Rightarrow X^s = 1 \text{ and } X^+ = X \tag{3.39}$$
$$X \leqslant 0 \Rightarrow X^s = 0 \text{ and } X^+ = 0 \tag{3.40}$$

Proposition 3.6 (Linearization for operators "+" and "s"). *Let X be one variable. Let α be the* sign *expression X^s. Let β be the* positive *exprssion X^+. Let $X \in \mathbb{Z}$ and $B \in \mathbb{N}^+$ be* "sufficiently large". *Let $\alpha \in \{0, 1\}$ and $\beta \in \mathbb{N}$.*

There exists 5 *equations to compute α and β:*

$$\begin{cases} B \cdot \alpha - X & \leqslant & B - 1 \\ X - B \cdot \alpha & \leqslant & 0 \\ X - \beta & \leqslant & 0 \\ \beta - B \cdot \alpha & \leqslant & 0 \\ \beta + B \cdot \alpha - X & \leqslant & B \end{cases} \quad s.t. \quad \begin{cases} X > 0 & \Leftrightarrow & \alpha & = & 1 \\ & & \wedge \ \beta & = & X \\ X \leqslant 0 & \Leftrightarrow & \alpha & = & 0 \\ & & \wedge \ \beta & = & 0 \end{cases}$$

Using additional variables and constraints, it is thus possible to propose a general linear mathematical model to solve the Timed PNs reachability problem.

The full model denoted as $IP(\kappa)$ is given in figure (3.4). Equations (3.41) to (3.44) correspond to conditions over initial and final states. Equations (3.45) to (3.51) express the constraints over discrimination variables used to compute the $"+"$ and $"s"$ operators. Variables (a_k), (α_k) and (β_k) denote respectively the values of $\overrightarrow{s_{r_k}}^s$, $\left(\overrightarrow{s_{r_k}} - \Delta_{v_k} \cdot \overrightarrow{Id}\right)^s$ and $\left(\overrightarrow{s_{r_k}} - \Delta_{v_k} \cdot \overrightarrow{Id}\right)^+$ from equations (3.37) and (3.38). Equations (3.53) and (3.54) correspond to intermediate state computation equations (3.37) and (3.38). Equation (3.52) is used to forbid reentrant steps: if transition j is already active at date v_k (ie. $\alpha_{(k-1)j} = 1$), it cannot be fired again at the same date (ie. σ_{kj} cannot be equal to 1). Finally, equations (3.55) to (3.64) define improvements and the domain of variables.

3.4.2 Comparison between CP Model and IP Model

We have used two Timed PNs reachability problems to verify the efficiency and the robustness of the CP model with comparison to the IP model, as our paper [HBYT12b]. We choose to use the search strategy – labeling σ_k and s_{m_k} with value ordering from high to low and Δ_k and s_{r_k} with value ordering from low to high using step-by-step strategy.

The first TPN corresponds to the scheduling problem given in Figure (3.2). When searching for all solutions, $\overrightarrow{s_{m_0'}} = [1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1]$ is set as initial marking and $\overrightarrow{s_{m_f'}} = [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1]$ is set as final marking to simplify the search.

The second one is given in Figure (3.5). It represents philosophers around a table, who spend time eating spaghetti and thinking. To eat, each one needs two forks, but there is only one available for two people. Each philosopher is provided with a control place, allowing us to quantify how many times it has been active. The presence of this unbounded place makes the corresponding reachability graph unbounded.

The corresponding reachability problem is to find firing sequences allowing to make each philosopher eat a given number of times (2). Such problem is characterized by the existence of *deadlocks*, i.e. states from where no transition is fireable. Such situation can be obtained if each philosopher decide to take the

Let (R, d) be a TPN with its initial state s_0 given at date v_0. Let s_f be a target state. The constraint programming model $IP(\mathsf{K})$ is defined by

$$\forall k \in [\![0, \mathsf{K}-1]\!], \forall i \in [\![1, \mathrm{M}]\!], \forall j \in [\![1, \mathrm{N}]\!]$$

initial information

$$s_{m0i} = m_0(i) \tag{3.41}$$
$$s_{m\mathsf{K}i} = m_f(i) \tag{3.42}$$
$$s_{r0j} = r_0(j) \tag{3.43}$$
$$s_{r\mathsf{K}j} = r_f(j) \tag{3.44}$$

update residual duration vector

$$B \cdot a_{kj} - s_{rkj} \leqslant B - 1 \tag{3.45}$$
$$s_{rkj} - B \cdot a_{kj} \leqslant 0 \tag{3.46}$$
$$B \cdot \alpha_{kj} - s_{rkj} + \Delta_k \leqslant B - 1 \tag{3.47}$$
$$s_{rkj} - \Delta_k - B \cdot \alpha_{kj} \leqslant 0 \tag{3.48}$$
$$s_{rkj} - \Delta_k - \beta_{kj} \leqslant 0 \tag{3.49}$$
$$\beta_{kj} - B \cdot \alpha_{kj} \leqslant 0 \tag{3.50}$$
$$\beta_{kj} + B \cdot \alpha_{kj} - s_{rkj} + \Delta_k \leqslant B \tag{3.51}$$
$$\sigma_{(k+1)j} + \alpha_{kj} \leqslant 1 \tag{3.52}$$
$$s_{r(k+1)j} - \beta_{kj} = d_j \cdot \sigma_{kj} \tag{3.53}$$

update marking

$$s_{m(k+1)i} - s_{mki} = \sum_{c=0}^{\mathrm{N}-1} C_{ic}^{+} \cdot (a_{kc} - \alpha_{kc}) - \sum_{c=0}^{\mathrm{N}-1} C_{ic}^{-} \cdot \sigma_{(k+1)c} \tag{3.54}$$

modeling improvements and initial domains

$$\sum_{j=0}^{\mathrm{N}-1} \sigma_{(k+1)j} \neq 0 \tag{3.55}$$
$$\Delta_{v_k} \leqslant \max_{j \in [\![1,\mathrm{N}]\!]} s_{rkj} \tag{3.56}$$
$$s_{r(k+1)j} \leqslant d(j) \tag{3.57}$$
$$\sigma_{kj} \in \{0, 1\} \tag{3.58}$$
$$a_{kj} \in \{0, 1\} \tag{3.59}$$
$$\alpha_{kj} \in \{0, 1\} \tag{3.60}$$
$$\beta_{kj} \in [\![0, D_{max}]\!] \tag{3.61}$$
$$s_{rkj} \in [\![0, D_{max}]\!] \tag{3.62}$$
$$s_{mik} \in [\![0, M_{max}]\!] \tag{3.63}$$
$$\Delta_{v_k} \in]\!]0, D_{max}]\!] \cup \Delta_{v_0} \in [\![0, D_{max}]\!] \tag{3.64}$$

Figure 3.4: Integer Programming Model

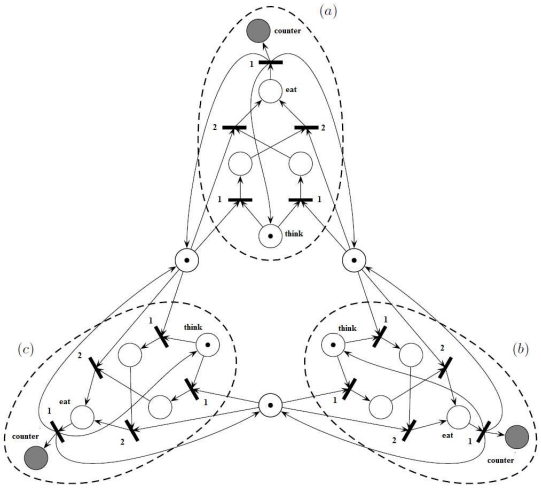


Figure 3.5: Dining Philosophers Problem

fork on his left, for example. The presence of *deadlocks* make the problem harder to solve.

As can be seen in tables (3.8) and (3.9), our conditional model behaves better than the IP one for the first example, particularly when the search space is larger. CP model is more efficient than IP model although they are meeting the same number of fails when searching with the same number of firing dates. Note that in table (3.9), for comparing the efficiency of CP model and IP model, the strategy is to label σ_k, s_{m_k}, Δ_k and s_{r_k} with value ordering from low to high using step-by-step strategy.

In table (3.10), the philosophers problem shows that although it meets far more fails during search CP model still search for solutions a little bit faster than IP model.

The use of conditional constraints allows us to reduce the number of variables and constraints of the model. One should note that the first solution given by each model using the default labeling strategy is the same: linearization variables were not chosen to be labeled in the search tree, they have been instantiated thanks to the enumeration of other variables.

With linearization, Solver will meet more variables in the search tree, but can control variables corresponding to (s_{r_k}) more conveniently. But as shown in section (3.3), variables (s_{r_k}) do not make much efficiency in searching for a solution. Therefore, CP model is more efficient than IP model in benchmarks of

K	Time	Fails	Choice points	Vars	Nb of solutions
CP model					
6	0 s	36	68	333	33
7	0.13 s	567	2045	381	1479
8	1.01 s	3884	27163	429	23280
9	6.61 s	17525	182612	477	165088
10	22.51 s	72673	610960	525	538288
11	33.65 s	238690	895213	573	656524
12	15.60 s	447419	447418	621	0
IP model					
6	0 s	36	68	503	33
7	0.14 s	567	2045	581	1479
8	1.31 s	3884	27163	659	23280
9	8.64 s	17525	182612	737	165088
10	29.44 s	72673	610960	815	538288
11	44.15 s	238690	895213	893	656524
12	21.25 s	447419	447418	971	0

Table 3.8: Scheduling Problem - Finding All Solutions

this section.

3.5 Conclusion

In this chapter, we have successfully translated our incremental model of TPNs into constraint programming model. In this process, the most important technique is to use conditional constraint to represent non-linear equations of firing sets. To search for one solution or all solutions more efficiently, we introduce the overall search strategies in CP under a example-driven tour. Then we point out the most efficient search strategy – labeling σ_k and s_{m_k} from high to low, labeling Δ_{k-1} and s_{m_r} from low to high using a dedicated *step-by-step* labeling strategy.

In the first section, we briefly introduce the terminology of CP. In conclusion, there are two main parts for modeling a CP model: *Modeling* and *Solving*. In the *Modeling* part, we describe a CSP and model it with variables and constraint to express relations between variables. In the *Solving* part, a *search tree* is generated at the same time of searching for solutions. Solver will branch the nodes of search tree by selecting variables with *variable ordering* and generate the leaves of search tree by choosing values of present variable (node) using *value ordering*.

CP model					
K	Time	Fails	variables	constraints	makespan
9	27.160 s	545210	477		no solution
10	8.486 s	171494	525	885	30
11	1.950 s	39341	573	974	30
12	0.577 s	10817	621	1057	34
13	0.172 s	1944	669	1147	34
14	0.125 s	1076	717	1235	38
15	0.047 s	77	765	1317	40
16	0.078 s	31	813	1408	41
17	0.031 s	26	861	1500	41
IP model					
9	33.821 s	532375	737		no solution
10	10.483 s	167508	815	1179	30
11	2.465 s	38646	893	1297	30
12	0.686 s	10650	971	1415	34
13	0.156 s	1919	1049	1533	34
14	0.094 s	1062	1127	1651	38
15	0.109 s	78	1205	1769	40
16	0.031 s	28	1283	1887	41
17	0.062 s	20	1361	2005	41

Table 3.9: Scheduling Problem - Finding First Feasible Solution

CP model					
K	Time	Fails	variables	constraints	Nb of solutions
13	1.856 s	22084	907	0	0
14	6.224 s	76286	972	0	2880
15	19.797 s	222456	1037	0	47328
16	55.458 s	435438	1102	0	291552
IP model					
13	1.716 s	17054	1477	0	0
14	6.006 s	56522	1587	0	2880
15	20.374 s	165514	1697	0	47328
16	61.808 s	356742	1807	0	291552

Table 3.10: Dining Philosophers Problem - Finding All Solutions

Each time, one variable is instantiated, Solver propagates constraints to reduce the search tree until a solution is found. If no solution (or a deadend) is found, Solver will backtrack to other open node using backtrack search algorithm. The most popular algorithms will be using Depth-First Search backtracking search

algorithm and keep arc consistency (constraint propagation) during search.

In the second section, the incremental model is translated into CP model following the process introduced in the first section. The conditional constraint is used to express the non-linear equations of firing sets, which is very strong technique to model non-linear equations in a natural way using the lowest number of variables. This advantage can reduce the search space and improve the efficiency. Then we give the basic benchmarks of CP model using a scheduling problem example to present the efficiency of our model.

In the third section, we give benchmarks on different search strategies. With the structure information, labeling variables σ_k and s_{m_k} using *value ordering* from high to low, and labeling variables Δ_{k-1} and s_{r_k} using a *step-by-step* search strategy is developed to search for solution, which are proved to be the most efficient. To use search strategies under a controlled way, DFS is chosen as the backtrack search algorithm.

Finally, a linearization technique is applied to control variables corresponding to s_{r_k} in CP model. After comparing the CP model and IP model, CP model seems more efficiency than IP model. Although linearization technique brings up more variables and controlled variables s_{r_k} in a search strategy, it cannot bring much improvements as shown in section (3.3).

To conclude, we have developed an incremental model and efficient search strategies in CP. In the next chapter, we will apply the incremental model and CP techniques to manufacturing systems.

Chapter 4

Application of the Incremental Approach for the Reconfiguration of Manufacturing Systems

The objective of this chapter is to show how an incremental approach can be applied to the reconfiguration of manufacturing systems and show its efficiency. Many manufacturing systems are time critical systems in the real world. In this thesis we are interested in finding reconfiguring actions, which can be expressed as a timed firing sequence.

In the first section, the main issues of manufacturing systems are first introduced using a *reconfigurable transport system*. The reconfiguration process consists in two stages: *decision-making* and *operational process*. The decision-making part consists in determining for a plant, a new objective state to reach. The operational process consists to determine the *procedure to apply* to reach this objective state from the current faulty state.

Then, the methodology for modeling systems is introduced based on TPNs. The TPNs model is main constituted of *Pregraph* and *Extended Operating Sequence*. They describe respectively the routing alternatives and the whole combinations of machining operations to produce a finished part. In addition, additional models are introduced to perform the reconfiguration of the plant.

Tokens for producing different productions can be confused in the *Pregraph*. Therefore in the subsection (4.1.3), token confusion issues are described using

a simple example. Then token identification techniques for ordinary PN are introduced following [HBT07]. We define an extended new *vector of identifiers*, where a non negative integer is associated to every place containing token, to denote the identity of the token in the place. The non negative integer number in one place is called token *ID*. This token identification technique can be well used for safe PN.

In the second section, we adapt this token identification technique to TPNs with safe behavior. Since tokens are missing during one transition firing, a new vector $\overrightarrow{s_{Rd}}$ is defined to record the corresponding missing token identifiers. Transporting token identifiers is more complex using TPNs than ordinary PN, since token identifiers are moved from places to transitions at the beginning of one transition firing and from transitions to places at the end of this firing. This method is limited to TPNs with safe behavior, since two token identifiers may be confused in the same place.

Thus, in the third section, we develop the *token identification technique for bounded TPNs*. As explained previously, token identification technique is limited to safe nets. Meanwhile the process of partitioning and updating token identifiers in TPNs are complex. Thus, we propose to develop one token identifier vector for each kind of token (tokens with the same *ID* number), $\overrightarrow{s_{Id_x}}$ and $\overrightarrow{s_{Rd_x}}$ being the corresponding vectors. This improvements help implement token identification technique in *bounded TPNs*. Each token identifiers vector $\overrightarrow{s_{Id_x}}$ can be seen as a marking in the TPNs with D as the incidence matrix. Different token identifiers vector only interact with each other based on transitions in the original TPNs with traditional incidence matrix. This method can reduce the complexity with partitioning and updating token identifiers between places and transitions, since each token identifiers vector seems to be updated independently in its own net. Thus, we define this partial independence nets as *token identifier layer in TPNs*.

Since each token identifier layer in TPNs seems to be independent in its own net, *Token identification technique for bounded TPNs* can easily express the behavior of TPNs with different kinds of tokens. Although the search space seems to be enlarged for the token identifier layer in TPNs, token identifiers must follow the marking of original TPNs. Therefore the efficiency of searching for a solution is nearly the same as the original TPNs.

Then, we compare those token identification techniques using the same exam-

ple to show the efficiency of our methods. But when studying the reconfigurable transport system, we found out that solutions are hardly found due to loops in *Pregraph*, since they lead the solver to go deeply in the search tree.

Therefore in the fourth section, we propose two kinds of methods to avoid the influence of loops based on token identifier nets in TPNs. First, conditional constraints based on the structure of *Pregraph* are introduced to avoid loops of robots. Even if this method can well lead solver to get solutions, this method is very specific to this model. Second, we develop one new vector – firing priority $\overrightarrow{FP}$ – for transitions in *Pregraph*. The element of $\overrightarrow{FP}$ shows the priority (high number with high priority) of one transition and will be set as 0 after firing. Therefore tokens cannot fire the same transition again, which can help avoiding loops. Since $\overrightarrow{FP}$ is defined based on each token identifier, tokens with different *ID* will not be forbidden for the same transition. If tokens need to go into some loops, $\overrightarrow{FP}$ can be reset according to these needs.

4.1 Reconfiguration of Manufacturing Systems

In this section, the practical manufacturing system – reconfigurable transport system – is introduced to validate all our approaches in this thesis.

4.1.1 Illustrative Example

The concept of *reconfiguration process* has been introduced first in [Ber98] as a solution to react to fault. We introduce a small plant instance that will be used to illustrate our methodology.

The reconfiguration process is a control function that aims to reorganize the structure (from the point of view of plant resources) in order to verify the user requirements with regards to production, and to minimize the number of resources in production. In this context, two reconfiguration situations can be distinguished [PBTZ05].

- *When starting a new production.* In this situation some resources that were used in the previous production horizon are no more required for the new production. These resources must be stopped. On the contrary, other resources must be started to enable the new production.

- *When a resource breaks down.* In this case one must react quickly to continue the production without using the broken resource. The general method consists in replacing the set of resources associated only to the broken resource. [Ber98] proposes to distinguish three sub-cases starting from a minor reconfiguration (use of previously activated resources) till a major reconfiguration (use of resources that were previously inactive).

To illustrate our approach, we consider the example of figure (4.1). This workshop is made of *four machines-tools*: two *turning machines* (T_1 and T_3) and two *machining machines* (M_2 and M_4). The machining operations of each machine are given above its representation. Parts are loaded (or unloaded) on these machines by *four robots*. The arrows in this figure represent the reachability capacity of each robot.

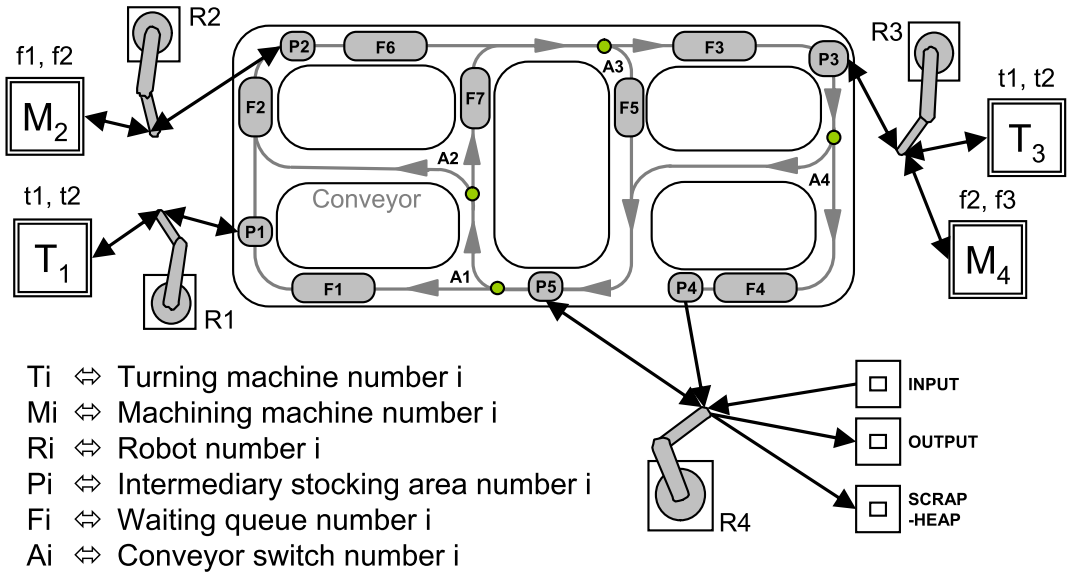


Figure 4.1: An Example of FMS with a Reconfigurable Transport System [Ber98]

This workshop offers both *structural redundancy* with the two turning machines (T_1 and T_3), and *functional redundancies* with operations (e.g. f_2) that can be executed by several machines. (e.g. M_2 or M_4).

A *belt conveyor* enables a flexible routing of the parts between the machines. The parts are transported on this conveyor on *pallets*. The conveyor is made of loading/unloading areas denoted by P_i that have each a capacity of one pallet.

Before or after these areas, *FIFO waiting queues* are available, denoted by F_i. Each of them have a capacity of several pallets. They allow stocking temporary pallets when a loading/unloading area is not free. Pallets are routed in different directions by the use of *conveyor switches* denoted by A_i. For this study and for simplicity, we assume that the switches are the only components of the conveyor that can be broke down. The other components that can fail are machines-tools and robots.

4.1.2 Reconfiguration Methodology based on Timed Petri Nets

The assigned objective is to reuse the models that have been developed for the control, in order to determine the actions needed to perform the reconfiguration of the plant:

- The *Pregraph* which is an aggregated model that describes the routing alternatives;
- The *Extended Operating Sequence* which describes the whole combinations of machining operations allowing to produce a finished part.

These two models are based on TPN formalism. To associate time durations for transitions, machines take 25 time units to complete the operations, robots take 9 time units, conveyor switches takes 5 time units and F_i, P_i take 3 time units. Others are taking 1 time unit. We explain the main elements of the considered model below.

4.1.2.1 Transport System

The Pregraph (figure (4.2)) is a TPN model that represents the feasible transfers between the different locations of the transport system. In figure (4.2), each grey place represents a physical area such as a buffer or a machine. The transitions between these places model the possibilities of transfer between physical areas in direct reachability. Critical resources like robots are represented by one token in the corresponding place marking.

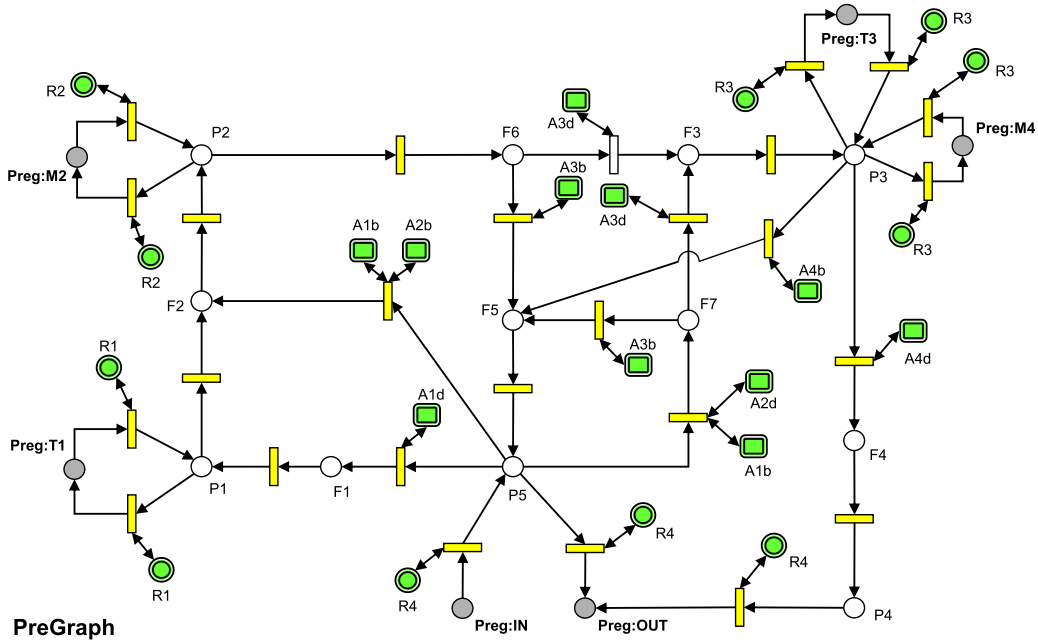


Figure 4.2: The Pregraph of figure (4.1)

.

4.1.2.2 Extended Operating Sequences

An Extended Operating Sequence (EOS) describes the different ways to obtain a *finished product* from *raw parts* using the available machines of a plant. An EOS is formally modeled as a TPN where places model the *location* of a part with regards to *characteristic areas**, and transitions model *operations* on this part. Operations can be machining operations or transport operations between two characteristic areas.

Two EOS OS_1 (t_1, f_2) and OS_2 (f_3, t_2) are presented in figure (4.3). For example, producing products with OS_2. Since the plant works as a job shop and t_2 can be held by T_1 and T_3, the corresponding two combinations are represented by two branches in the TPNs model.

In an EOS, white color places are used to represent the status of a part in the

*A physical area is a characteristic area if it is a working area (machining or assembling area) or if it is an area that enables the transfer of a part from the considered resource to another one.

system. For example, the status $t1-$ means that the operation $T1$ is planned, but has not yet been done. The status $t1+$ means that this operation has been performed. The status as $T1 \rightarrow M4$ means that the part is transported from area $T1$ to area $M4$.

Note that we have represented several times the same place to make the figure more readable.

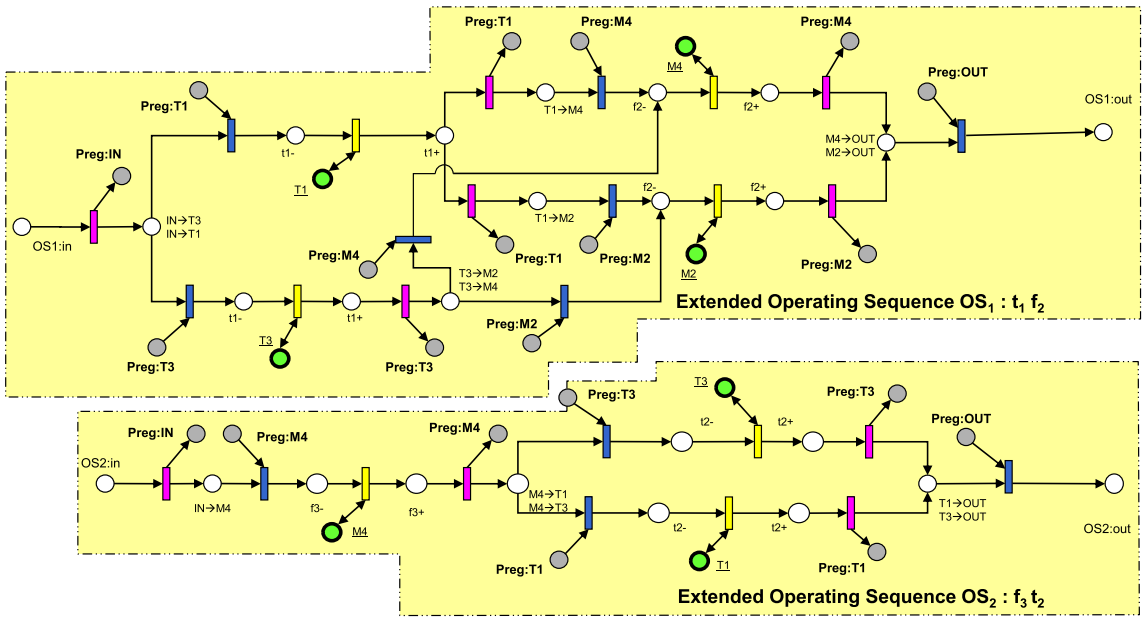


Figure 4.3: Operating Sequences of figure (4.1)

.

4.1.2.3 Conveyor Switches

The representation of switches is presented in figure (4.4). Depending on its position, a switch enables or not the firing of a transition of the model. The switch model illustrates that two transfer transitions that use the same switch in different positions cannot be fired at the same time. The model used for conveyor switches allows also to model the failure of a switch. Indeed, one has just to remove the resource token on the inner place, like for A_2, to model a blocked switch in position A_{2d}.

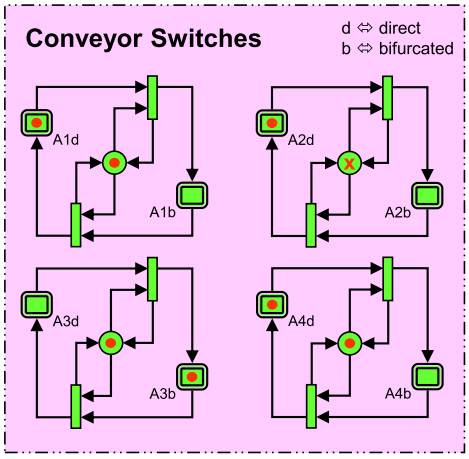


Figure 4.4: Conveyor Switches of figure (4.1)

.

4.1.2.4 Reconfiguration Actions

We propose to add for each resource $\mathcal{R}$ that must be handled by the reconfiguration process two places and two transitions as shown in figure (4.5). The meaning of these elements is as follows. The left place $\mathcal{R}_{i\text{-off}}$ means that resource $\mathcal{R}_i$ is stopped. The inner place is a *resource* that indicates whether the state of $\mathcal{R}_i$ *can* be changed or not. If the inner place is marked, the transitions *"on"* and *"off"* can be used to switch the resource $\mathcal{R}_i$ between working and suspending mode.

Three situations can be modeled using such a technique, depending on the marking of the inner place. On the given example, the robot R_1 is working, but can be stopped if it is no more necessary. Then R_1 off. Finally, if there is no token in the inner place of R_1, it cannot be run anymore.

Since possible reconfiguration actions have been represented directly in the reconfiguration model, a *succesfull search* for a firing sequence leading to a final marking in which each operating sequence is *finished* will allow to *deduce* the needed reconfiguration actions.

The model is finally refined in order to reduce the search space and allow the identification of operating sequences that *cannot be achieved* considering the remaining available resources.

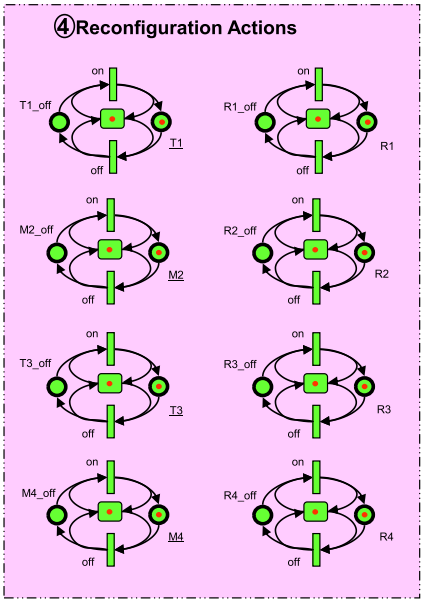


Figure 4.5: Reconfiguration Actions of figure (4.1)

.

4.1.2.5 Full TPN Model

In figure (4.6), we give the full TPN model for representing the reconfiguration example of figure (4.1).

One has to merge all places that have the same name to get the true TPNs model. The same technique is used for all the remaining models below.

Together, *Pregraph* and *Extended Operating Sequence* models hold implicitly the *whole information* regarding the reconfiguration process. These models are *refined* to make explicitly appear the reconfiguration actions, in order to be able to solve them using our CP approach. This refined model is called the *"reconfiguration"* model.

Therefore, two reconfiguration situations can be expressed as reachability problem as bellow:

- *When starting a new production.* One token is added to each OS_1*:in* as the initial marking. For final marking, 2 tokens should be got in the *Target* place and 1 token for each OS_i*:out* place. In this case, this final marking is the target marking state to be reached with regard to reachability problem. All

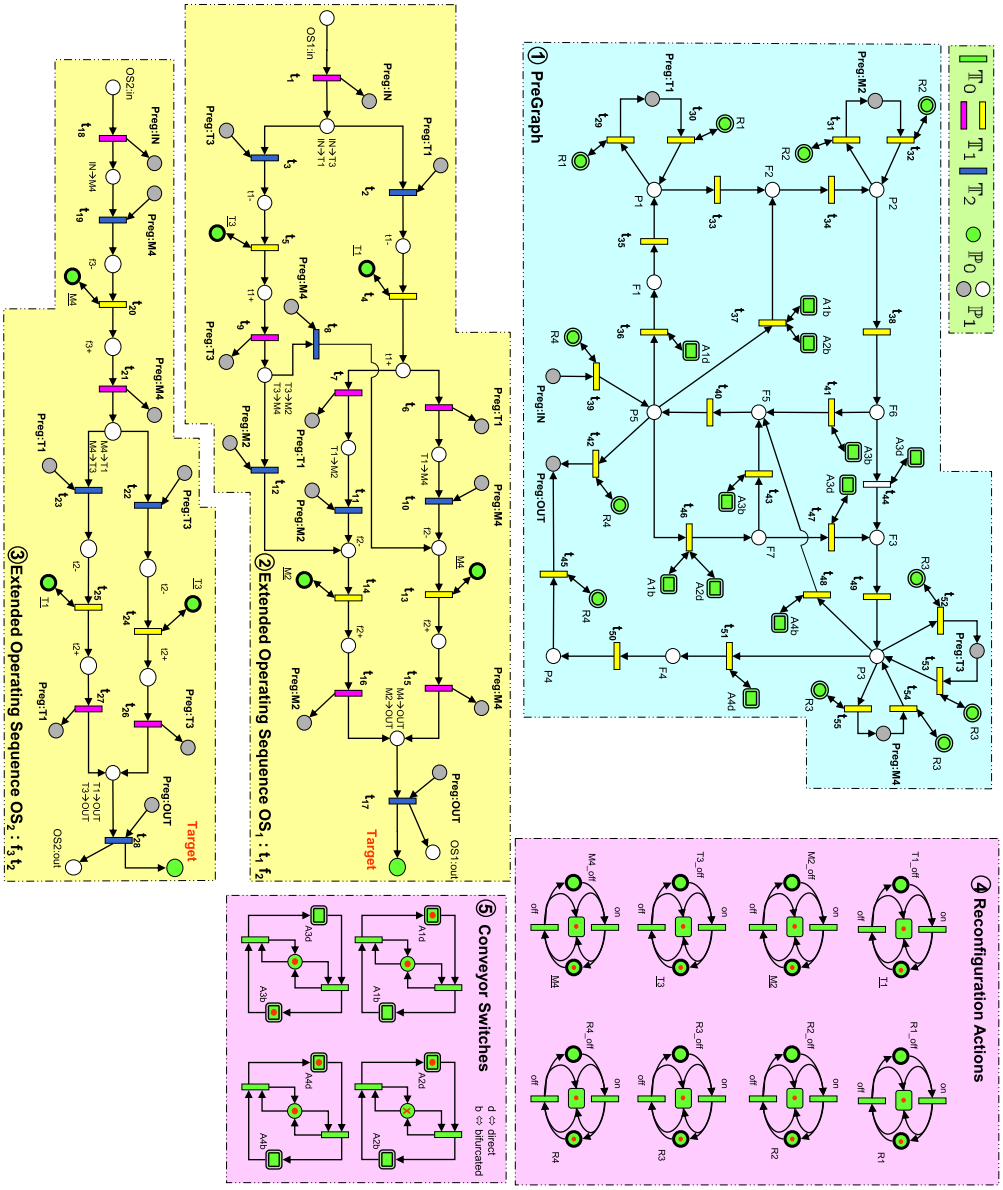


Figure 4.6: PN Reconfiguration Model for the Example of figure (4.1)

.

solutions will present all the possibility of operating sequences to produce products and corresponding reconfiguration actions.

- *When a resource breaks down.* The initial marking can be somewhere in operating sequences in figure (4.6). We assume that places OS_1*:in* and

OS_2:*in* are taking 1 token at the beginning. The final marking still has 2 tokens in the *Target* place and 1 token for each OS_i:*out* place. All possible firing sequences will be given regardless of whether one resource is active or broken down.

Under these two situations, if all possible firing sequences are resolved with our approaches, reconfiguration actions can be easily given. For example, if one resource is broken down, the possible firing sequence will tell the corresponding actions to finish the same product.

4.1.3 Token Identification

TPN is a good modeling framework to express the behavior of DES, such as transport or manufacturing systems. They allow to represent easily the distribution of tasks within a complex system, with the capacity to handle time constraints on the duration of these tasks.

However, when dealing with *real-world systems* like manufacturing or transport systems, where different parts can cross the system and reach the same location, synchronization is needed between several layers of the system. Under this situation, TPN models suffer from a lack of mechanisms to identify the components that are handled. Thus, it is difficult to obtain firing sequences avoiding confusion of tokens when activation or coordination patterns are used.

Such issues are well addressed with High Level PNs, like Colored PNs where data structures are attached to tokens, allowing to represent a complex behavior while keeping a compact PN model. However, such models introduce a modeling material that makes it difficult to use *direct* incremental solving approaches like mathematical or constraint programming, i.e. approaches that do not need to unfold the net and build its whole reachability graph.

In this thesis, we propose to follow the methods proposed in [HBT07] and [BT09] to enhance the expressivity of our modeling methodology by associating *identifiers* to tokens – expressed as positive integers – and defining new fireability conditions allowing to synchronize tokens and identificators. Such approach can avoid the confusion of parts in a plant, or trains in a railway network, and allow to model much more possible coordination situations. A constraint programming

model is built to follow this new firing semantics, and benchmarks are given to assess its efficiency.

4.1.3.1 Token Confusion Issues

The figure (4.6) is the considered TPN model of figure (4.1), two EOS can execute their tasks at the same time, which may lead to load the *Pregraph* with two tokens at the same moment, representing parts going through the transport system from one machining location to another.

Since two tokens are transported in the same *Pregraph*, some spurious solutions can be found if one cannot distinguish tokens coming from the *Pregraph*. Indeed: since tokens in the grey places in figure (4.6) can go into both Operating Sequences, this situation will mix up these two tokens and can lead the system to get a false production.

For example, one token (for OS_2) has finished the operation of M_4 and come to p_1. Another token (for OS_1) has just come to p_5. The first token may go into OS_1 and continue to produce its corresponding product. Then the second token will continue to T_1, but it will produce the product OS_2. However the first token will take operations of three machines and the second token will take only one operation of T_1. Such conditions will produce two defective products.

One way to fix this problem is to protect the use of *Pregraph* using *semaphore* mechanisms, but this led to an increase of the number of steps needed to reach the target marking, and thus the practical complexity of the resolution. Another attempt was tried where each EOS was linked to its own *Pregraph*. Results show that in the general case, the resolution takes more time with 2 Pregraphs than with only one, since the PN is larger.

4.1.3.2 Token Identification for Ordinary Petri Nets

To enhance the expressivity of our modeling methodology and strengthen its ability to model real-world reconfiguration problems, we follow the methods proposed in [HBT07] to handle the confusion problem by associating *identifiers* to some tokens in an *PN model*, and by modifying the token game accordingly.

Definition 4.1 (TPN State Extended with Token Identifiers). *Let* (R, d) *be a TPN. Its* state *can be extended as* $s_k = (\overrightarrow{s_{m_k}}, \overrightarrow{s_{r_k}}, \overrightarrow{s_{Id_k}})$, *where* $k \in [\![0, K]\!]$,

$\overrightarrow{s_{Id_k}} \in \mathbb{N}^{\mathsf{M}}$ *is the vector of identifiers for distinguishing tokens. A non negative integer is associated to some places of the PN, to denote the identity of the token in this place after the* k^{th} *step firing.*

This additional state vector is used to *constrain* the fireability conditions of transitions, and extended reachability equations are defined to ensure that the tokens and their identifiers move synchronically in the PN.

Note that we assume the considered PN is *non weighted*, i.e. $C^+, C^- \in \{0,1\}^{\mathsf{M}\times\mathsf{N}}$.

In order to minimize the practical complexity of the global formulation, we restrict the use of identifiers to the places of the net for which it brings *effective advantages*. Thus, we define formally 2 sets of places ($\mathbb{P}_0$, $\mathbb{P}_1$) – associated or not with the vector of identifiers – and 3 sets of transitions ($\mathbb{T}_0$, $\mathbb{T}_1$, $\mathbb{T}_2$) – according to the structure of their relations with the surrounding places – and give the corresponding behavioral equations. These sets define respectively a partition of $\mathbb{P}$ and $\mathbb{T}$.

A small example of such partitions is given in figure (4.7) and its corresponding matrices are given in figure (4.8).

In [HBT07], the token identification technique has the restriction of safe behavior, since if two tokens and theire corresponding token identifiers can be moved to the same place, token identifiers will be mix up.

The reason for this restriction is that one element of token identifier vector $\overrightarrow{s_{Id_k}}$ can just represent one token identifier. For example, if the firing sequence t_2, t_1, t_3 is happened, token identifiers $ID = 1$ and $ID = 2$ will be moved to the same place p_7. Then $\overrightarrow{s_{Id_k}}(p_7) = 1 + 2 = 3$. Then when t_4 is fired, this token identifier will be moved to place p_8 with one token. This situation will lead to the confusion of token identifiers as presented in table (4.1).

Therefore, equation (4.1) is given to forbid this kind of firings in order to keep PN safe. Therefore, we restrict the study to search for firing sequences of length K that keep the PN in a *safe state* (at most one token in each place), i.e. that respect the constraint:

$$\forall k \in [\![1, \mathsf{K}]\!], \forall p \in \mathbb{P}, s_{m_k}(p) \leqslant 1 \tag{4.1}$$

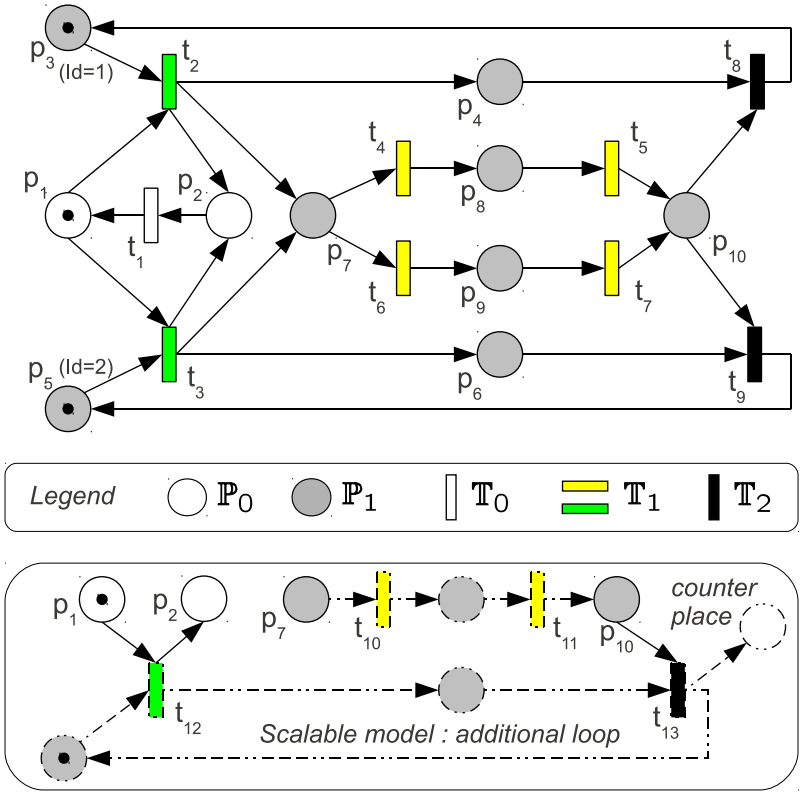


Figure 4.7: A PN with Token Identification.

Using such restriction, token identifiers can be handled by a structure associated to places. However, this limitation will reduce the expressivity of PN and TPN model. Thus in section (4.3), we will propose a token identification technique for bounded TPNs.

$\mathbb{P}_0, \mathbb{T}_0$ – Classical Places and Transitions

When dealing with *resources places* which denote only the binary availability of a machine, robot, switch or section, we consider *no identification extension* and continue to use *classical PNs equations*. The corresponding places and transitions are colored in white in figure (4.7). This set of places is not constrained to contain safe markings.

In the following, each set of transitions is suffixed by the number of upstream places linked with this transitions for which it is necessary to handle a token

C	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
p_1	1	-1	-1	·	·	·	·	·	·
p_2	-1	1	1	·	·	·	·	·	·
p_3	·	-1	·	·	·	·	·	1	·
p_4	·	1	·	·	·	·	·	-1	·
p_5	·	·	-1	·	·	·	·	·	1
p_6	·	·	1	·	·	·	·	·	-1
p_7	·	1	1	-1	·	-1	·	·	·
p_8	·	·	·	1	-1	·	·	·	·
p_9	·	·	·	·	·	1	-1	·	·
p_{10}	·	·	·	·	1	·	1	-1	-1

Figure 4.8: Incidence Matrix for the PN of figure (4.7)

Firing	Marking	$\overrightarrow{s_{Id}}$	$\overrightarrow{\Theta^{j}_{k+1}}$
initial	$(1,0,1,0,1,0,0,0,0,0)$	$(0,0,1,0,2,0,0,0,0,0)$	$\overrightarrow{0_M}$
t_2	$(0,1,0,1,1,0,1,0,0,0)$	$(0,0,0,1,2,0,1,0,0,0)$	$(0,0,-1,1,0,0,1,0,0,0)$
t_1	$(1,0,0,1,1,0,1,0,0,0)$	$(0,0,0,1,2,0,1,0,0,0)$	$\overrightarrow{0_M}$
t_3	$(0,1,0,1,0,1,2,0,0,0)$	$(0,0,0,1,0,2,3,0,0,0)$	$(0,0,0,0,-2,2,2,0,0,0)$
t_4	$(1,0,0,1,0,1,0,1,0,0)$	$(0,0,0,1,0,2,0,3,0,0)$	$(0,0,0,0,0,0,-3,3,0,0)$

Table 4.1: The Behavior of Firing Sequence t_2, t_1, t_3 of figure (4.7)

identifier.

$\mathbb{P}_1$ – Extended Places

A place belongs to $\mathbb{P}_1$ if it is associated to the vector of identifiers. The corresponding places are colored in gray in figure (4.7).

The set $\mathbb{P}_1$ is used to define a new data structure $D \in \{-1,0,1\}^{\mathsf{M}\times\mathsf{N}}$ corresponding to the restriction of the incidence matrix to the extended places and the transitions that do not belong to $\mathbb{T}_0$. Formally, $\forall p_i \in \mathbb{P}, \forall t_j \in \mathbb{T}, D_{ij} = C_{ij}, D^-_{ij} = C^-_{ij}, D^+_{ij} = C^+_{ij}$ if $p_i \in \mathbb{P}_1$ and $t_j \notin \mathbb{T}_0$, and 0 otherwise, as is shown in figure (4.9). Therefore $\forall p_i \in \mathbb{P}, \forall t_j \in \mathbb{T}, D_{ij} = D^+_{ij} - D^-_{ij}$.

$\mathbb{T}_1$ – Forwarding Transitions

A transition t_j belong to this set if it is linked with exactly *one* $\mathbb{P}_1$ *upstream place*, i.e. if $|\{p_i \in \mathbb{P}_1 \; s.t. \; D^-_{ij} = 1\}| = 1$. In figure (4.7), we have colored in yellow the

C	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
p_1	1	−1	−1	·	·	·	·	·	·
p_2	−1	1	1	·	·	·	·	·	·
p_3	·	−1	·	·	·	·	·	1	·
p_4	·	1	·	·	·	·	·	−1	·
p_5	·	·	−1	·	·	·	·	·	1
p_6	·	·	1	·	·	·	·	·	−1
p_7	·	1	1	−1	·	−1	·	·	·
p_8	·	·	·	1	−1	·	·	·	·
p_9	·	·	·	·	·	1	−1	·	·
p_{10}	·	·	·	·	1	·	1	−1	−1
D	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
p_1	·	·	·	·	·	·	·	·	·
p_2	·	·	·	·	·	·	·	·	·
p_3	·	−1	·	·	·	·	·	1	·
p_4	·	1	·	·	·	·	·	−1	·
p_5	·	·	−1	·	·	·	·	·	1
p_6	·	·	1	·	·	·	·	·	−1
p_7	·	1	1	−1	·	−1	·	·	·
p_8	·	·	·	1	−1	·	·	·	·
p_9	·	·	·	·	·	1	−1	·	·
p_{10}	·	·	·	·	1	·	1	−1	−1
$u^1_{t_j}$	·	3	5	7	8	7	9	4	6
$u^2_{t_j}$	·	·	·	·	·	·	·	10	10

Figure 4.9: The Partition Data Structures for the PN of figure (4.7)

transitions t_4, t_5, t_6, t_7 belonging to $\mathbb{T}_1$ which correspond to operations of transfer of identifiers from one place to another. Such transitions are associated with only *one downstream place belonging to* $\mathbb{P}_1$: $|\{p_i \in \mathbb{P}_1 \ s.t. \ D^+_{ij} = 1\}| = 1$.

Transitions t_2 and t_3 colored in green correspond to *logical duplications* of the token identifier between two branches of the PN (e.g. firing t_2, identifier from p_3 goes to p_4 and p_7). Such transitions are associated with *two downstream places belonging to* $\mathbb{P}_1$: $|\{p_i \in \mathbb{P}_1 \ s.t. \ D^+_{ij} = 1\}| = 2$.

The previous definition allows associating to each transition t from $\mathbb{T}_1$ the unique index $u^1_t \in [\![1, \mathsf{M}]\!]$ (also denoted by u^1_j if $t \equiv t_j$) of the upstream place of t belonging to $\mathbb{P}_1$. For example, $u^1_{t_2} = 3$ represents that p_3 is the only one upstream place, which can contain token identifiers

This index is used to formulate the equations given below, that will be used to express the update of $\overrightarrow{s_{Id}}$ after the firing of step $\overrightarrow{\sigma_{k+1}}$.

Let $t_j \in \mathbb{T}_1$. We define the vector $\overrightarrow{\Theta^j_{k+1}} \in \mathbb{Z}^{\mathsf{M}}$ such that:

$$\begin{cases} \sigma_{(k+1)j} = 1 & \Rightarrow \quad \overrightarrow{\Theta^j_{k+1}} = Id_k(u^1_j) \cdot \overrightarrow{D_j} \\ \sigma_{(k+1)j} = 0 & \Rightarrow \quad \overrightarrow{\Theta^j_{k+1}} = \overrightarrow{0} \end{cases} \tag{4.2}$$

where $\overrightarrow{D_j}$ stands for the column vector of D corresponding to t_j. Using the definition of $\mathbb{T}_1$ and table (4.1), one can note that $\overrightarrow{\Theta^j_{k+1}}$ contains exactly *one negative value*, and may contain several positive ones, *all of them being equal* in absolute value – since the net is *non weighted* – to the identifier that was associated to place $p_{w^1_j}$ *before the step* σ_{k+1} *was fired*. For instance, firing t_2 at step $k+1$ when p_3 (since $u^1_{t_2} = 3$) contains a token carrying identifier <*i*> leads to $\overrightarrow{\Theta^2_{k+1}} = -$ <*i*> $\cdot\overrightarrow{e_{p_3}} +$ <*i*> $\cdot(\overrightarrow{e_{p_4}} + \overrightarrow{e_{p_7}})$.

Thus, if one sums $\overrightarrow{\Theta^j_{k+1}}$ to the previous vector of identifiers $\overrightarrow{s_{Id_k}}$, the result would correspond to the transfer of an identifier from the extended upstream place of t_j to its extended downstream places.

By *iterating* this operation to the whole set of transitions belonging to $\mathbb{T}_1$, one makes the vector of identifiers being updated in the desired way. The corresponding equation can finally be expressed in the following way, after the firing of step σ_{k+1}:

$$\overrightarrow{s_{Id_{k+1}}} = \overrightarrow{s_{Id_k}} + \sum_{t_j \in \mathbb{T}_1} \overrightarrow{\Theta^j_{k+1}} \tag{4.3}$$

$\mathbb{T}_2$ – Synchronization Transitions

A transition t_j belongs to this set when it is linked to at last *two extended upstream places*, i.e. when $|\{p_i \in \mathbb{P}\ s.t.\ D^-_{ij} = 1\}| = 2$. The transitions are colored in black in figure (4.7). As above, we denote by u^1_t and u^2_t (or u^1_j, u^2_j) the indexes of the extended upstream places of t_j. For instance $u^1_{t_9} = 6, u^2_{t_9} = 10$. It means that p_6 and p_{10} are extended upstream places of t_9, which can take token identifiers.

This set $\mathbb{T}_2$ is used to synchronize tokens with same token identifiers, since synchronize transitions, like t_8 may be fired by tokens in upstream places with different token identifiers.

Take table (4.2) as an example, after t_5 firing, both t_8 and t_9 are enabled. If t_9 is fired in the ordinary PN, token identifiers are updated by equation (4.2).

Firing	Marking	$\overrightarrow{s_{Id}}$	$\overrightarrow{\Theta^{j}_{k+1}}$
initial	$(1,0,1,0,1,0,0,0,0,0)$	$(0,0,1,0,2,0,0,0,0,0)$	$\overrightarrow{0_M}$
t_2	$(0,1,0,1,1,0,1,0,0,0)$	$(0,0,0,1,2,0,1,0,0,0)$	$(0,0,-1,1,0,0,1,0,0,0)$
t_1	$(1,0,0,1,1,0,1,0,0,0)$	$(0,0,0,1,2,0,1,0,0,0)$	$\overrightarrow{0_M}$
t_4	$(1,0,0,1,1,0,0,1,0,0)$	$(0,0,0,1,2,0,0,1,0,0)$	$(0,0,0,0,0,0,-1,1,0,0)$
t_3	$(0,1,0,1,0,1,1,1,0,0)$	$(0,0,0,1,0,2,2,1,0,0)$	$(0,0,0,0,-2,2,2,0,0,0)$
t_5	$(0,1,0,1,0,1,1,0,0,1)$	$(0,0,0,1,0,2,2,0,0,1)$	$(0,0,0,0,0,0,0,-1,0,1)$
t_9	$(0,1,0,1,1,0,1,0,0,0)$	$(0,0,0,1,2,0,2,0,0,-1)$	$(0,0,0,0,2,-2,0,0,0,-2)$

Table 4.2: Synchronization of Token Identifiers of figure (4.7)

Since $u^1_{t_9} = 6$, token identifers are removed as $ID = 2$, which leave $ID = -1$ for p_{10}. This situation will mix up tokens, and t_8 cannot be fired again.

For that, we define an *additional fireability condition* associated to each transition t_j in $\mathbb{T}_2$, in order to verify that the identifiers from $p_{u^1_j}$ and $p_{u^2_j}$ are equal when firing σ_{k+1}:

$$\forall t_j \in \mathbb{T}_2, \overrightarrow{\sigma_{k+1}}(j) = 1 \Rightarrow \overrightarrow{s_{Id_k}}(u^1_j) = \overrightarrow{s_{Id_k}}(u^2_j) \tag{4.4}$$

The update equation (4.2) used previously must also be used with the transitions belonging to $\mathbb{T}_2$, since the equality $\overrightarrow{s_{Id_k}}(u^1_j) = \overrightarrow{s_{Id_k}}(u^2_j)$ holds: the identifiers associated with places $p_{u^1_j}$ and $p_{u^2_j}$ will move to the extended downstream places.

Above equations led to an implementation in [HBT07], to address the problem of reconfiguration of manufacturing systems, using ordinary PN. Some linearization techniques were proposed to adapt their formulation to the context of a linear integer programming.

4.2 Adaptation of Token Identification to Timed Petri Nets with Safe Behavior

In this section, we propose to directly adapt the token identification technique for ordinary PN for TPNs. However, this token identification for TPNs still has the limitation of keeping safe behavior.

Firing	Date	$\overrightarrow{s_m}$	$\overrightarrow{s_r}$	$\overrightarrow{s_{Id}}$	$\overrightarrow{\Theta^J_{k+1}}$
initial	0	$(1,0,1,0,1,0,0,0,0,0)$	$\overrightarrow{0_N}$	$(0,0,1,0,2,0,0,0,0,0)$	$\overrightarrow{0_M}$
t_2	0	$(0,0,0,0,1,0,0,0,0,0)$	$s_r(t_2)=\delta$	$(0,0,0,0,2,0,0,0,0,0)$	$(0,0,-1,1,0,0,1,0,0,0)$
	δ^-	$(0,1,0,1,1,0,1,0,0,0)$	$\overrightarrow{0_N}$	$(0,0,0,1,2,0,1,0,0,0)$	$\overrightarrow{0_M}$
t_1	δ	$(0,0,0,1,1,0,1,0,0,0)$	$s_r(t_1)=\delta$	$(0,0,0,1,2,0,1,0,0,0)$	$\overrightarrow{0_M}$
	$\delta+\delta^-$	$(1,0,0,1,1,0,1,0,0,0)$	$\overrightarrow{0_N}$	$(0,0,0,1,2,0,1,0,0,0)$	$\overrightarrow{0_M}$
t_3	$2\cdot\delta$	$(0,0,0,1,0,0,1,0,0,0)$	$s_r(t_3)=\delta$	$(0,0,0,1,0,0,1,0,0,0)$	$(0,0,0,0,-2,2,2,0,0,0)$
	$3\cdot\delta$	$(0,1,0,1,0,1,2,0,0,0)$	$\overrightarrow{0_N}$	$(0,0,0,1,0,2,3,0,0,0)$	$\overrightarrow{0_M}$
t_4	$3\cdot\delta+1$	$(0,1,0,1,0,1,1,0,0,0)$	$s_r(t_4)=\delta$	$(0,0,0,1,0,2,0,0,0,0)$	$(0,0,0,0,0,0,-3,3,0,0)$

Table 4.3: Token identifers without Safe Behavior Restriction for TPNs of figure (4.7)

4.2.1 Informal Presentation

The main issue when adapting token identification structure to TPN is to maintain the association between tokens and identifiers, since TPN semantics make some tokens "disappear" when transitions are fired. To illustrate this issue, we consider the PN of figure (4.7), where all transitions are given an arbitrary duration δ. Let us consider the firing sequence $(t_2,0)$, (t_1,δ), $(t_3,2\cdot\delta)$, $(t_4,3\cdot\delta+1)$. Two questions hold regarding place p_7.

4.2.2 Ensuring Safe Behavior

As shown in table (4.3), what happens if transition t_4 is not fired before the end of t_3 firing ? At date $3\cdot\delta$, transition t_3 finishes and p_7 contains 2 tokens, which is forbidden by our *safety hypothesis*, since vector $\overrightarrow{s_{Id}}$ can only store one identifier by place.

When dealing with ordinary PN, the equation (4.1) set on each intermediate marking ensures this situation cannot happen. Considering TPN, delays induced in the marking vector updates after a firing make difficult to impose such constraint in a simple way. It is not sufficient to constrain only *at each firing date*, that the marking must remains safe. Indeed: a transient marking not safe may appear before two firing dates and be covered by the next firing, making the reached marking being safe.

This is the case for instance if we consider the timed sequence $(t_2,0)$, (t_1,δ), $(t_3,2\cdot\delta)$, $(t_4,3\cdot\delta+1)$. When t_3 is fired at date $k=2\cdot\delta$, the marking is still safe since $\overrightarrow{s_{m_k}}=(0,0,0,1,0,0,1,0,0)^\intercal$. When t_3 ends at date $k'=3\cdot\delta$, its downstream

places receive tokens and the marking becomes $\overrightarrow{s_{m_{k'}}} = (0,1,0,1,0,1,2,0,0)^\intercal$, which is not safe.

However, since the methodology described in section (2.2) updates states vectors only at firing dates (which is sufficient to ensure no place marking is negative and guarantees steps are fireable), the next date of updating marking will be only made when t_4 is fired, at date $3 \cdot \delta + 1$. At this date, one token of p_7 is consumed by t_4, so the marking remains safe *at first sight*. Such situation cannot be accepted since there is an uncertainty about which identifier must be used to fire t_4.

In order to fix this issue, without checking markings at any date which would reveal inefficient, we propose to split equation (2.9), updating markings between dates v_k and v_{k+1}, into two parts.

First, tokens resulting from finishing transitions are added to the previous marking $\overrightarrow{s_{m_k}}$, in order to verify the resulting marking is still safe by equation (4.5):

$$\overrightarrow{s_{m_k}} + \sum_{t_j \in T^f_{k+1}} C^+ \cdot \overrightarrow{e_{t_j}} \leqslant \overrightarrow{1} \tag{4.5}$$

Since no tokens can be removed by this operation, this ensures all intermediate marking between dates v_k and v_{k+1} are safe. This equation replaces equation (4.1).

Then, tokens consumed by the firing of $\overrightarrow{\sigma_{k+1}}$ are removed to produce marking $\overrightarrow{s_{m_{k+1}}}$, as stated in equation (2.9).

4.2.3 Storing Identifiers for Firing Transitions

Let us now consider the firing sequence $(t_2, 0)$, (t_1, δ), $(t_3, 2 \cdot \delta)$, $(t_4, 2 \cdot \delta + 1)$ as shown in table (4.4). Place p_7 is associated to identifier 1 between dates δ (when t_2 finishes) and $2 \cdot \delta + 1$ (when t_4 is fired). Thus, when t_4 finishes at date $3 \cdot \delta + 1$, place p_8 should be associated with identifier 1. At date $k = 3 \cdot \delta$, the firing of transition t_3 ends. A token is created in place p_7, and identifier 2 is moved to $s_{Id_k}(p_7)$. Since transition t_4 is not finished at this date, where should be stored the identifier 1 that must be moved in p_8 at date $3 \cdot \delta + 1$? In fact, the $\overrightarrow{\Theta^J_{k+1}}$ are working in a wrong way to represent token identifiers.

Obviously, identifiers must not be stored in vector $\overrightarrow{s_{Id}}$ during the time a transition is firing. Thus in our paper [HBYT12a], we propose to add new additional vector of identifiers, $\overrightarrow{s_{Rd}} \in \mathbb{N}^{|\mathbb{T}_1, \mathbb{T}_2|}$, which collects identifiers used by currently

Firing	Date	$\overrightarrow{s_m}$	$\overrightarrow{s_r}$	$\overrightarrow{s_{Id}}$	$\overrightarrow{s_{Rd}}$	$\overrightarrow{\Theta^j_{k+1}}$
initial	0	$(1,0,1,0,1,0,0,0,0,0)$	$\overrightarrow{0_N}$	$(0,0,1,0,2,0,0,0,0,0)$	$\overrightarrow{0_N}$	$\overrightarrow{0_M}$
t_2	0	$(0,0,0,0,1,0,0,0,0,0)$	$s_r(t_2)=\delta$	$(0,0,0,0,2,0,0,0,0,0)$	$s_{Rd}(t_2)=1$	$(0,0,-1,1,0,0,1,0,0,0)$
	δ^-	$(0,1,0,1,1,0,1,0,0,0)$	$\overrightarrow{0_N}$	$(0,0,0,1,2,0,1,0,0,0)$	$\overrightarrow{0_N}$	$\overrightarrow{0_M}$
t_1	δ	$(0,0,0,1,1,0,1,0,0,0)$	$s_r(t_1)=\delta$	$(0,0,0,1,2,0,1,0,0,0)$	$\overrightarrow{0_N}$	$\overrightarrow{0_M}$
	$\delta+\delta^-$	$(1,0,0,1,1,0,1,0,0,0)$	$\overrightarrow{0_N}$	$(0,0,0,1,2,0,1,0,0,0)$	$\overrightarrow{0_N}$	$\overrightarrow{0_M}$
t_3	$2\cdot\delta$	$(0,0,0,1,0,0,1,0,0,0)$	$s_r(t_3)=\delta$	$(0,0,0,1,0,0,1,0,0,0)$	$s_{Rd}(t_3)=2$	$(0,0,0,0,-2,2,2,0,0,0)$
t_4	$2\cdot\delta+1$	$(0,0,0,1,0,0,0,0,0,0)$	$s_r(t_4)=\delta$ $s_r(t_3)=\delta-1$	$(0,0,0,1,0,0,0,0,0,0)$	$s_{Rd}(t_4)=1$	$(0,0,0,0,0,0,-1,1,0,0)$
	$3\cdot\delta$	$(0,1,0,1,0,1,1,0,0,0)$	$s_r(t_4)=1$	$(0,0,0,1,0,2,2,0,0,0)$	$s_{Rd}(t_4)=1$	$\overrightarrow{0_M}$
	$3\cdot\delta+1$	$(0,1,0,1,0,1,1,1,0,0)$	$\overrightarrow{0_N}$	$(0,0,0,1,0,2,2,1,0,0)$	$\overrightarrow{0_N}$	$\overrightarrow{0_M}$

Table 4.4: Token identifers in TPNs of figure (4.7)

firing transitions. To use and update $\overrightarrow{s_{Rd}}$, we proceed in the same way as in the above subsection.

First, identifiers *resulting from finishing transitions* are *added* to the previous identifiers vector $\overrightarrow{s_{Id_k}}$ to build an intermediate vector $\overrightarrow{s_{Id}}^{aux}_{k+1}$.

Let T^f_{k+1} denotes the set of transitions finishing between steps σ_k and σ_{k+1}. Equation (4.2) is replaced by:

$$\forall t_j \in T^f_{k+1}, \quad \overrightarrow{\Phi^j_{k+1}} = s_{Rd_{kj}} \cdot \overrightarrow{D^+_j} \tag{4.6}$$

$$\forall t_j \in \mathbb{T}_1 \cup \mathbb{T}_2 - T^f_{k+1}, \quad \overrightarrow{\Phi^j_{k+1}} = \overrightarrow{0} \tag{4.7}$$

Equation (4.3) is used as above to produce $\overrightarrow{s_{Id}}^{aux}_{k+1}$ from $\overrightarrow{s_{Id_k}}$ and $(\overrightarrow{\Phi^j_{k+1}})_{t_j\in\mathbb{T}_1\cup\mathbb{T}_2}$. This intermediate vector is used to check synchronization for $\mathbb{T}_2$ transitions in a similar way as equation (4.4).

Then, when firing σ_{k+1}, $\overrightarrow{s_{Id_{k+1}}}$ and $\overrightarrow{s_{Rd_{k+1}}}$ are produced in the following way:

$$\forall t_j \in \mathbb{T}_1 \cup \mathbb{T}_2,$$

$$\sigma_{(k+1)j} = 1 \;\Rightarrow\; \begin{cases} s_{Rd_{(k+1)j}} = s_{Id^{aux}_{k+1}}(u^1_j) \\ \overrightarrow{\Psi^j_{k+1}} = s_{Id^{aux}_{k+1}}(u^1_j) \cdot \overrightarrow{D^-_j} \end{cases} \tag{4.8}$$

$$\sigma_{(k+1)j}) = 0 \;\Rightarrow\; \begin{cases} s_{Rd_{(k+1)j}} = s_{Rd_{kj}} \\ \overrightarrow{\Psi^j_{k+1}} = \overrightarrow{0} \end{cases} \tag{4.9}$$

Again, equation (4.3) is used as above to produce $\overrightarrow{s_{Id_{k+1}}}$ from $\overrightarrow{s_{Id}}^{aux}_{k+1}$ and $(\overrightarrow{\Psi^j_{k+1}})_{t_j\in\mathbb{T}_1\cup\mathbb{T}_2}$. Note identifiers stored in $\overrightarrow{s_{Rd}}$ are never removed even when a transition finishes: if the same transition is used again, the current identifier will

replace the previous one thanks to equation (4.8).

Working like that, identifiers produced by transitions finishing before step $\overrightarrow{\sigma_{k+1}}$ can be *reused* by transitions belonging to $\overrightarrow{\sigma_{k+1}}$. Identifiers *"disappear"* from $\overrightarrow{s_{Id_k}}$, are *"stored"* in $\overrightarrow{s_{Rd_k}}$ when transitions are fired, then *"move back"* to $\overrightarrow{s_{Id_{k+1}}}$ when transitions finish, just before $\overrightarrow{\sigma_{k+1}}$ is fired.

4.2.4 Constraint Programming Formulation

A CP model without token identifiers has been given in figure (3.3) of chapter (3). We just add new defined variables and constraints between them into the CP model. The new defined variables are:

- Identifiers vector $\overrightarrow{s_{Id_k}} \in \mathbb{N}^{|\mathbb{P}_1|}$
- Residual identifiers vector $\overrightarrow{s_{Rd_k}} \in \mathbb{N}^{|\mathbb{T}_1 \cup \mathbb{T}_2|}$

Since equations (4.2) to (4.9) are developed following the incremental approach, we can easily translate them into CP formulations with conditional constraints.

Since the full model is too big and has just been developed from the basic CP model (figure 3.3) by adding token identifiers, only equations corresponding to token identifiers are added in figure (4.10). Reader can merge these two figures to see the full model. Equations (4.10) to (4.13) define the initial and target token identifiers information. Equations (4.14) and (4.20) are used to update identifiers vectors $\overrightarrow{s_{Id_{k+1}}}$ and $\overrightarrow{s_{Rd_{k+1}}}$. Equation (4.21) to (4.24) give the domain of variables of token identifiers. ID_{max} denotes the maximum identifier number used in the considered problem.

4.3 Token identification for Bounded Timed Petri Nets

The token identifiers model can be well used in TPNs with safe behavior. However in the real world, it should be more general, for example each place can contain several tokens with different token identifiers. One token identifier vector

Let $R = (\mathbb{P}, \mathbb{T}, C^+, C^-, d, \{\overrightarrow{s_{m_0}}, \overrightarrow{s_{r_0}}\})$ be a TPN. Let $\mathbb{P}_0, \mathbb{P}_1$ and $\mathbb{T}_0, \mathbb{T}_1, \mathbb{T}_2$ be the partitions of $\mathbb{P}$ and $\mathbb{T}$ defined above. For each transition t_j in $\mathbb{T}_1$ or $\mathbb{T}_2$, let u_j^1, u_j^2 be the respective places indexes defined above. Let $D \in \{-1, 0, 1\}^{\mathsf{M} \times \mathsf{N}}$ be the extended incidence matrix defined above. Let $\{Id_0, Rd_0\}$ be its initial vectors of identifiers. Let $\{m_f, r_f, Id_f, Rd_f\}$ be a target state. Let $\mathsf{K} \in \mathbb{N}$ be the number of considered firing dates. The constraint programming model $CPID(\mathsf{K})$ is defined by $CP(\mathsf{K})$ and token identifiers equations below:

$$\forall k \in [\![0, \mathsf{K}-1]\!], \forall i \in [\![1, \mathrm{M}]\!], \forall j \in [\![1, \mathrm{N}]\!]$$

initial information

$$s_{Id_{\mathsf{0}i}} = Id_{0i} \tag{4.10}$$
$$s_{Id_{\mathsf{K}i}} = Id_{fi} \tag{4.11}$$
$$s_{Rd_{\mathsf{0}j}} = Rd_{0j} \tag{4.12}$$
$$s_{Rd_{\mathsf{K}j}} = Rd_{fj} \tag{4.13}$$

update residual duration vector

update marking

update ID

$$t_j \in \mathbb{T}_1 \cup \mathbb{T}_2, T^f_{(k+1)_j} = 1 \Rightarrow \Phi_{(k+1)ji} = s_{Rd_{kj}} \cdot D^+_{ij} \tag{4.14}$$
$$t_j \in \mathbb{T}_1 \cup \mathbb{T}_2, T^f_{(k+1)_j} = 0 \Rightarrow \Phi_{(k+1)ji} = 0 \tag{4.15}$$
$$s_{Id^{aux}_{(k+1)i}} = s_{Id_{ki}} + \sum_{t_j \in \mathbb{T}_1 \cup \mathbb{T}_2} \Phi_{(k+1)ji} \tag{4.16}$$
$$t_j \in \mathbb{T}_1 \cup \mathbb{T}_2, \sigma_{(k+1)_j} = 1 \Rightarrow s_{Rd_{(k+1)j}} = s_{Id^{aux}_{(k+1)u^1_j}} \wedge \quad \Psi_{(k+1)ji} = s_{Id^{aux}_{(k+1)u^1_j}} \cdot D^-_{ij} \tag{4.17}$$
$$t_j \in \mathbb{T}_1 \cup \mathbb{T}_2, \sigma_{(k+1)_j} = 0 \Rightarrow s_{Rd_{(k+1)j}} = s_{Rd_{kj}} \wedge \quad \Psi_{(k+1)ji} = 0 \tag{4.18}$$
$$s_{Id_{(k+1)i}} = s_{Id^{aux}_{(k+1)i}} + \sum_{t_j \in \mathbb{T}_1 \cup \mathbb{T}_2} \Psi_{(k+1)ji} \tag{4.19}$$
$$t_j \in \mathbb{T}_2, \qquad \sigma_{(k+1)_j} = 1 \Rightarrow s_{Id^{aux}_{(k+1)u^1_j}} = s_{Id^{aux}_{(k+1)u^2_j}} \tag{4.20}$$

modeling improvements and initial domains

$$s_{m_{ki}} \leqslant 1 - \sum_{c=1}^{\mathrm{N}} C^+_{ic} \cdot T^f_{(k+1)_c} \tag{4.21}$$
$$s_{Id_{(k+1)i}} \in [\![0, ID_{max}]\!] \tag{4.22}$$
$$s_{Rd_{(k+1)j}} \in [\![0, ID_{max}]\!] \tag{4.23}$$
$$s_{Id^{aux}_{(k+1)i}} \in [\![0, ID_{max}]\!] \tag{4.24}$$

Figure 4.10: Constraint Programming Model with Token Identifiers

$\overrightarrow{s_{Id}}$ cannot contain several different token identifiers, since they will be summed together in the same place. This situation will mix up token identifiers.

In this thesis, we propose to associate token identifier vectors $\overrightarrow{s_{Id_x}}$ and $\overrightarrow{s_{Rd_x}}$ to each kind of tokens in TPNs, where $x \in [\![1, X_{id}]\!]$, X_{id} being equal to the number of token identifiers. The main idea is to load all tokens with the same token identifier into one token identifier vector.

To illustrate our method, the figure (4.11) is given, which is derived from figure (4.7). The key idea is that each kind of token identifier vectors $\overrightarrow{s_{Id_x}}$ and $\overrightarrow{s_{Rd_x}}$ can be seen as a new TPN with the extended incidence matrix D and the same time duration $\overrightarrow{d}$. Indeed, $\overrightarrow{s_{Id_x}}$ corresponds to $\overrightarrow{s_m}$, $\overrightarrow{s_{Rd_x}}$ corresponds to $\overrightarrow{s_r}$ and D corresponds to C. In this thesis, we call them *token identifier layer in TPNs* and associate them to each kind of tokens. Each token identifier vectors $\overrightarrow{s_{Id_x}}$ and $\overrightarrow{s_{Rd_x}}$ can be computed as state of TPNs.

As been seen, there are three TPNs in this example, where one original TPN and two *token identifier layers*. Therefore, each token identifier layer can be updated as follow:

$$\forall k \in [\![0, \mathsf{K}-1]\!], \forall i \in [\![1, \mathrm{M}]\!], \forall j \in [\![1, \mathrm{N}]\!]$$

$$s_{Id_{x(k+1)i}} = s_{Id_{xki}} - \sum_{j=1}^{\mathrm{N}} D^-_{ij} \cdot \sigma_{x(k+1)j} + \sum_{j=1}^{\mathrm{N}} D^+_{ij} \cdot T^f_{x(k+1)j} \tag{4.25}$$

$$s_{Rd_{x(k+1)j}} = s_{Rd_{xkj}} + \sigma_{x(k+1)j} - T^f_{x(k+1)j} \tag{4.26}$$

$$x \in [\![1, X_{id}]\!] \tag{4.27}$$

$$s_{id_{xki}} \in [\![0, M_{max}]\!] \tag{4.28}$$

$$s_{Rd_{xkj}} \in \{0, 1\} \tag{4.29}$$

$$\sigma_{xkj} \in \{0, 1\} \tag{4.30}$$

$$T^f_{xkj} \in \{0, 1\} \tag{4.31}$$

The equation (4.25) is similar with updating marking vector in the equation (2.9). The number of tokens with the same token identifier in one place must be bounded with the M_{max} (the biggest number of tokens) in TPNs. In equation (4.26), $s_{Rd_{x(k+1)j}}$ is used to represent if transition t_j is fired or not, but not to record the ID number of token identifier as $s_{Rd_{(k+1)j}}$ in figure (4.10).

In equations (4.25 and 4.26), TPNs are not partitioned as in subsection (4.1.3), since the extended incidence matrix D will forbid token identifiers to go to $\mathbb{P}_0$

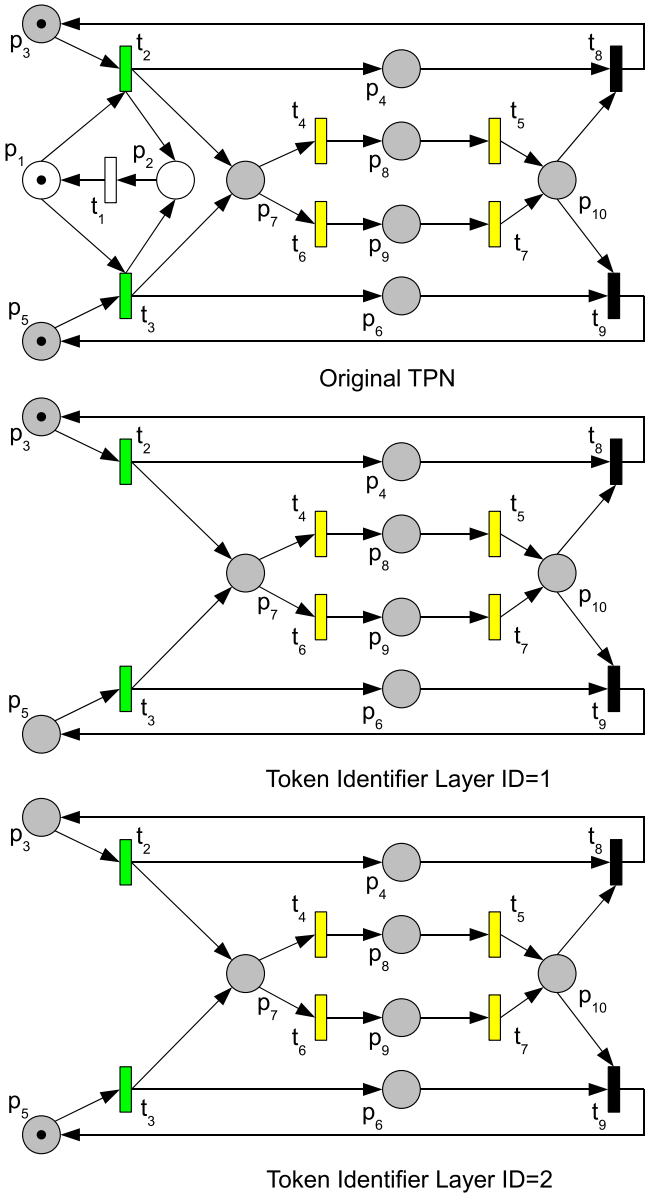


Figure 4.11: Token Identifiers for Bounded TPNs

and $\mathbb{T}_0$, as shown in section (4.2).

Then, the main problem is to accomplish that token identifiers must be synchonized with their corresponding tokens of the original TPN.

The ***mechanism*** of our approach is that if one transition t_j at date v is fired (or finished) in the original TPN, *if and only if* one of transitions t_j in different

Firing	Date	$\overrightarrow{s_m}$	$\overrightarrow{s_r}$	$\overrightarrow{s_{Id_x}}$	$\overrightarrow{s_{Rd_x}}$
initial	0	$(1,0,1,0,1,0,0,0,0,0)$	$\overrightarrow{0_N}$	$(0,0,1,0,0,0,0,0,0,0)$ $(0,0,0,0,1,0,0,0,0,0)$	$\overrightarrow{0_N}$ $\overrightarrow{0_N}$
t_2	0	$(0,0,0,0,1,0,0,0,0,0)$	$s_r(t_2)=\delta$	$(0,0,0,0,0,0,0,0,0,0)$ $(0,0,0,0,1,0,0,0,0,0)$	$s_{Rd_1}(t_2)=1$ $\overrightarrow{0_N}$
	δ^-	$(0,1,0,1,1,0,1,0,0,0)$	$\overrightarrow{0_N}$	$(0,0,0,1,0,0,1,0,0,0)$ $(0,0,0,0,1,0,0,0,0,0)$	$\overrightarrow{0_N}$ $\overrightarrow{0_N}$
t_1	δ	$(0,0,0,1,1,0,1,0,0,0)$	$s_r(t_1)=\delta$	$(0,0,0,1,0,0,1,0,0,0)$ $(0,0,0,0,1,0,0,0,0,0)$	$\overrightarrow{0_N}$ $\overrightarrow{0_N}$
	$\delta+\delta^-$	$(1,0,0,1,1,0,1,0,0,0)$	$\overrightarrow{0_N}$	$(0,0,0,1,0,0,1,0,0,0)$ $(0,0,0,0,1,0,0,0,0,0)$	$\overrightarrow{0_N}$ $\overrightarrow{0_N}$
t_3	$2\cdot\delta$	$(0,0,0,1,0,0,1,0,0,0)$	$s_r(t_3)=\delta$	$(0,0,0,1,0,0,1,0,0,0)$ $(0,0,0,0,0,0,0,0,0,0)$	$\overrightarrow{0_N}$ $s_{Rd_2}(t_2)=1$
	$3\dot{\delta}$	$(0,1,0,1,0,1,2,0,0,0)$	$\overrightarrow{0_N}$	$(0,0,0,1,0,0,1,0,0,0)$ $(0,0,0,0,0,1,1,0,0,0)$	$\overrightarrow{0_N}$ $\overrightarrow{0_N}$
t_4	$3\cdot\delta+1$	$(0,1,0,1,0,1,1,0,0,0)$	$s_r(t_4)=\delta$	$(0,0,0,1,0,0,0,0,0,0)$ $(0,0,0,0,0,1,1,0,0,0)$	$s_{Rd_1}(t_4)=1$ $\overrightarrow{0_N}$

Table 4.5: Token Identifiers for Bounded TPNs of figure (4.11)

token identifier layers will be fired (or finished) at the same date.

We take the table (4.5) as an example, which represents the behavior figure (4.11) with firing sequence $(t_2,0)$, (t_1,δ), $(t_3,2\cdot\delta)$, $(t_4,3\cdot\delta+1)$.

When the transition t_2 in the original TPN is fired at date 0, if and only if the transition t_2 in the token identifier layer $ID=1$ is fired at date 0. Then transitions t_3 in the original TPN and the token identifier layer $ID=2$ are fired the same date $2\cdot\delta$. These situations present that if the enabled transition is only related to one kind of token identifiers, this kind of token identifers will be synchronized with their correspoding tokens.

When transition t_4 will be fired at date $3\cdot\delta+1$, there are two tokens and two token identifiers in p_7. In the table (4.3), the token identification for TPNs with safe behavior will mistake these two token identifiers as one token identifier with $ID=3$, which will mix up token identifiers.

However, in the table (4.5), if t_4 is fired, if and only if one transition t_4 in different token identifier layers can be fired. Then solver will choose one of these two t_4 to be fired by default or developed search strategy, for example t_4 in the token identifier layer $ID=4$ is fired. As shown in table (4.5), the safe behavior

restriction in table (4.3) is resolved.

As the definition (2.11), the equation (4.32) represents that if transitions in one token identifier layer can be fired, it must have enough tokens of the same token identifiers to satisfy the firable condition for transitions in this token identifier layer. However, for the necessary condition, it depends on whether other token identifier layers are satisfying firable condition for the same transition. Then, solver will choose one active transition in one token identifier layer to be fired.

In fact, if actions of transitions $\overrightarrow{\sigma_{x(k+1)}}$ and $\overrightarrow{T^f_{x(k+1)}}$ are known, each of these *token identifier layer in TPNs* can be solved independently. Thus, equations (4.33 and 4.34) are proposed to synchronize these vectors with original TPNs vectors.

The equation (4.35) ensures that finished transitions must be active at last step in the same token identifier layer.

$$\begin{aligned} &\forall k \in [\![0, \mathsf{K}-1]\!], \forall i \in [\![1, \mathrm{M}]\!], \forall j \in [\![1, \mathrm{N}]\!] \\ D^- \cdot \overrightarrow{\sigma_{x(k+1)}} &\leqslant \overrightarrow{s_{Id_{xk}}} + \sum_{t_j \in T^f_{x(k+1)}} D^+ \cdot \overrightarrow{e_{t_j}} \end{aligned} \tag{4.32}$$

$$\sigma_{(k+1)j} = \sum_{x=1}^{X_{id}} \sigma_{x(k+1)j} \tag{4.33}$$

$$T^f_{(k+1)j} = \sum_{x=1}^{X_{id}} T^f_{x(k+1)j} \tag{4.34}$$

$$T^f_{x(k+1)j} = 1 \Rightarrow s_{Rd_{xkj}} = 1 \tag{4.35}$$

For one transition t_j, its domains of $\sigma_{x(k+1)j}$ and $T^f_{x(k+1)j}$ are restricted in $\{0, 1\}$ by equations (4.30 and 4.31). Thus equation (4.33) constrains that when one transition is fired in original TPNs, if and only if one of the same transitions in different token identifier layers can be fired. Afterwards, the problem of choosing one token and its token identifier vector is achieved, when several different kinds of tokens are active before the same transition. Equation (4.34) constrains that only one finished transition in different token identifier layers can be synchronized with finished transition in original TPNs. The equation (4.35) constrains that this finished transition must be active at last firing date in the same token identifier layer.

Note that equations (4.33, 4.34) have the reaction from transitions in token

identifier layer to the original TPN. For example, if all transitions t_j in different token identifier layers cannot be fired, the transition t_j in the original TPN cannot be fired, even if there are enough tokens in its upstream places. Such constraints bypass the equation (4.4), which is used to insure that this transition will not be fired by mixed token identifiers.

Since the full model is too big and has just been developed from the basic CP model (figure 3.3) by adding token identifiers layers, only equations corresponding to token identifiers are added in figure (4.12). Reader can merge these two figure to see the full model. Equations (4.36) to (4.39) define the initial and target token identifiers information. Equations (4.40) and (4.45) replaces equations (4.14) to (4.20) to implement the function of token identifiers in bounded TPNs. Equations (4.42) to (4.43) ensure the firable condition and finished transition active at last step. Equations (4.46) to (4.49) give domains of variables of token identifiers.

4.4 Numerical Experiments of Token Identifiers

Experiments were carried out using a 2.93 GHz Pentium with 4 Gb of RAM, using the constraint programming tool *Ilog Solver*. We used the TPN with token identifiers presented in figure (4.7), with $d(t) = (1, 2, 1, 3, 5, 2, 6, 2, 3, 4, 4, 1, 3)$. To make the model scalable, an additional loop was used, shown in dashed in the figure. Each loop was completed by a *"counter place"* allowing to count the number of times each cycle has been executed.

We assess in table (4.6) the influence of dealing with token identifiers in our TPN model, compared with the model without identification. As it can be seen, the number of variables and constraints brought by models with token identifiers are much bigger than the previous one. However, computational times do not seem to be too sensible to this increase of complexity. This could be explained by the fact that token identification techniques do not generate new paths in the underlying reachability graph of TPNs. Meanwhile, the increasing constraints have reduced the influence of increasing variables.

In fact, all variables and constraints can be seen as supplementary constraints for original TPNs, since many paths with mixing tokens will be forbidden. Although the number of variables of token identifiers in bounded TPNs is much bigger than in TPNs with safe behavior for the same K, token identifiers in bounded

Let $R = (\mathbb{P}, \mathbb{T}, C^+, C^-, d, \{\overrightarrow{s_{m_0}}, \overrightarrow{s_{r_0}}\})$ be a TPN. Let $\mathbb{P}_0, \mathbb{P}_1$ and $\mathbb{T}_0, \mathbb{T}_1, \mathbb{T}_2$ be the partitions of $\mathbb{P}$ and $\mathbb{T}$ and $D \in \{-1, 0, 1\}^{\mathsf{M}\times\mathsf{N}}$ be the extended incidence matrix defined above. Let $\{Id_{x0}, Rd_{x0}\}$ be the initial vectors of token identifier $ID = x$. Let $\{m_f, r_f\}$ be a target state, associated with $\{Idx_f, Rd_{xf}\}$ with each kind of token identifiers. Let $\mathsf{K} \in \mathbb{N}$ be the number of considered firing dates. The constraint programming model $CPIDX(\mathsf{K})$ is defined by $CP(\mathsf{K})$ and token identifiers layers below:

$$\forall k \in [\![0, \mathsf{K}-1]\!], \forall i \in [\![1, \mathrm{M}]\!], \forall j \in [\![1, \mathrm{N}]\!], \forall x \in [\![1, X_{id}]\!]$$

initial information

$$s_{Id_{x0i}} = Id_{x0i} \tag{4.36}$$
$$s_{Id_{x\mathsf{K}i}} = Id_{xfi} \tag{4.37}$$
$$s_{Rd_{x0j}} = Rd_{x0j} \tag{4.38}$$
$$s_{Rd_{x\mathsf{K}j}} = Rd_{xfj} \tag{4.39}$$

update residual duration vector

update marking

update ID

$$\forall t_j \in \mathbb{T}_1, \quad \sigma_{(k+1)j} = \sum_{x=1}^{X_{id}} \sigma_{x(k+1)j} \tag{4.40}$$
$$\forall t_j \in \mathbb{T}_1, \quad T^f_{(k+1)j} = \sum_{x=1}^{X_{id}} T^f_{x(k+1)j} \tag{4.41}$$
$$\forall t_j \in \mathbb{T}_1, \quad T^f_{x(k+1)j} = 1 \Rightarrow s_{Rd_{xkj}} = 1 \tag{4.42}$$

$$D^- \cdot \overrightarrow{\sigma_{x(k+1)}} \leqslant \overrightarrow{s_{Id_{xk}}} + \sum_{t_j \in T^f_{x(k+1)}} D^+ \cdot \overrightarrow{e_{t_j}} \tag{4.43}$$

$$s_{Id_{x(k+1)i}} - s_{Id_{xki}} = \sum_{j=1}^{\mathrm{N}} D^+_{ij} \cdot T^f_{x(k+1)j} - \sum_{j=1}^{\mathrm{N}} D^-_{ij} \cdot \sigma_{x(k+1)j} \tag{4.44}$$
$$s_{Rd_{x(k+1)j}} - s_{Rd_{xkj}} = \sigma_{x(k+1)j} - T^f_{x(k+1)j} \tag{4.45}$$

modeling improvements and initial domains

$$s_{Id_{x(k+1)i}} \in [\![0, M_{max}]\!] \tag{4.46}$$
$$s_{Rd_{x(k+1)j}} \in \{0, 1\} \tag{4.47}$$
$$\sigma_{x(k+1)j} \in \{0, 1\} \tag{4.48}$$
$$T^f_{x(k+1)j} \in \{0, 1\} \tag{4.49}$$

Figure 4.12: Token Identifiers for Bounded TPNs

TPNs need less time. This can be explained easily, since each kind of tokens have

K	Time	Fails	variables	constraints	makespan
TPNs without token identifiers					
22	17.88 s	469872	1312	2691	56
23	4.24 s	112489	1369	2813	56
24	0.92 s	25798	1426	2933	60
25	0.33 s	10190	1483	3057	60
30	0.08 s	2	1768	3665	70
31	0.06 s	4	1825	3788	70
32	> 6000 s	time exceeded			
Token identifiers in TPNs with safe behavior					
22	30.00 s	220608	5267	6695	56
23	8.47 s	66507	5502	6998	56
24	3.70 s	33764	5737	7301	60
25	9.17 s	69633	5972	7606	60
30	0.05 s	2	7147	9117	70
31	0.03 s	2	7382	9421	70
32	> 6000 s	time exceeded			
Token identifiers in bounded TPNs					
22	13.65 s	193137	7799	9496	57
23	3.28 s	46174	8142	9927	57
24	1.37 s	18701	8485	10358	61
30	0.08 s	2	10543	12943	70
31	0.05 s	2	10886	13375	70
32	> 6000 s	time exceeded			

Table 4.6: Results of Token Identification Techniques

their own token identifiers vectors, they only influence each other when synchronization happens. Therefore, token identifiers in bounded TPNs will meet less fails.

Note that token identifers in bounded TPNs increase the opportunity of concurrency, since token identifiers in TPNs with safe behavior will forbid two token identifiers to be in the same places.

To conclude, token identification techniques can be well addressed for solving token confusion issues. Above all, the token identification technique in bounded TPNs gives the most efficiency in this example.

When we apply token identification technique in bounded TPNs to reconfigurable transport system in figure (4.6), Solver can hardly find one solution for some reachability problems, since Solver is trapped in ***loops***. Thus, we propose two methods to avoid loops in a reconfigurable transport system.

4.5 Avoiding Loops in Models for Reconfiguration Systems

In figure (4.6), it is easy to see that producing plants with two families of parts: $OS_1(t_1, f_2)$ and $OS_2(f_3, t_2)$ can be seen as a reachability problem. Thus, we want to generate firing sequences for these two products. To simplify the explanation, we use $GOAL$ to represent the reachability problem, For $GOAL$, the initial marking is 1 token for each OS_i*:in*. The final marking is 1 token for each OS_i*:out* place and 2 tokens for the *Target* place

In fact, the search space of the reconfigurable transport system presented in figure (4.6) is much larger than all previous examples. Therefore, developed search strategies must lead Solver not go too deep in the search tree. In the *Pregraph*, there are many loops that can lead Solver go deep into the search tree. These loops cannot be avoided, since tokens will need go through some loops.

Figure (4.13) is a screenshot from the figure (4.6) to present the problem of loops in reconfigurable transport system. For example, places p_1, $Preg : T1$ and transitions t_{29}, t_{30} form a loop. In fact, this loop is the function of robot R_1, which will take tokens from p_1 to $Preg : T1$, then take them back to p_1 in order to implement the function of machine $T1$. Therefore, tokens must go through this kind of loops to finish the product. After this operation, tokens need go forward to other machines.

However, in *Ilog Solver*, transitions with lower index numbers will be first chosen to to be fired by default. It means t_{29} will be fired instead of t_{33} and tokens will be trapped in this loop.

Figure (4.14) presents the search tree with the loop of R_1 in figure (4.13). For example, if the search depth is k, the token will go repeatly in the loop until the upper bound k reached. Then it backtracks p_1 at lower depth and go to F_2. However, the rest of search steps are not enough to finish all operations. Therefore, it backtracks to p_1 at lower depth again. This is the reason for hardly finding a solution for $GOAL$.

In fact, the token just need to go through this loop once fo finish the operation $T1$. Therefore, some algorithms are needed to avoid loops under some *requirements*. The requirement in the reconfigurable transport system is to go through loops *once at a time*. It means tokens can go back to the same loop after other

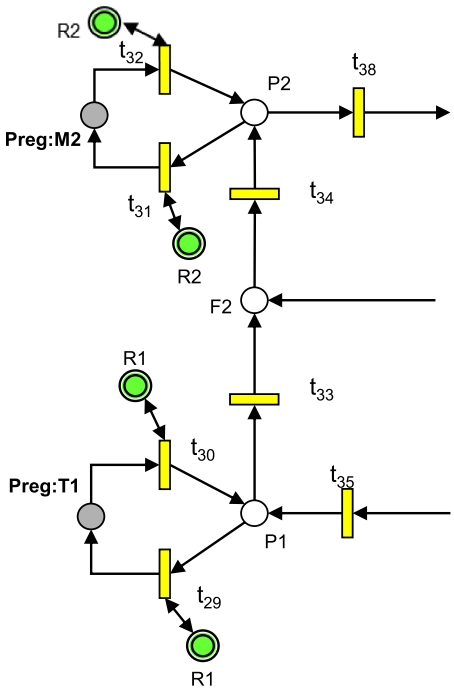


Figure 4.13: Problem of Loops in Reconfigurable Transport System

operations.

In this thesis, we propose two methods to avoid loops in reconfigurable transport system. The first method is developed based on the structure of TPN model. For example, loops formed by robots in figure (4.6) have been a main reason for leading Solver go deep in the search tree, we can avoid these loops directly. The second method is developed as general as possible. It applies a counter function. For example, if each transition can just be fired once, tokens cannot go to loops repeatedly.

4.5.1 Avoiding Loops of Robots in TPNs for CP

As been known that loops of robots will reduce the efficiency of searching, we propose to use some specific conditional constraints in the *Pregraph* to avoid loops of robots. We take the loop formed by R_1 as an example in figure (4.13). Since tokens are coming out of the loop, the start transition and the finish transition

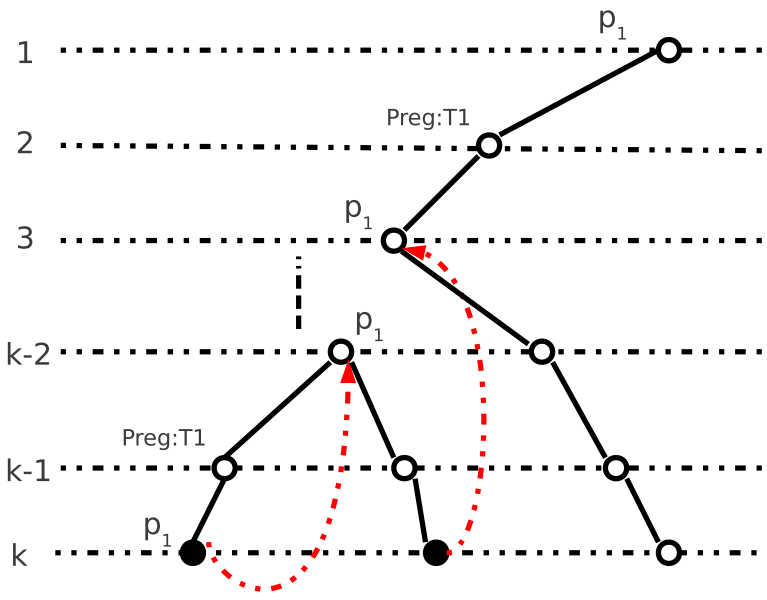


Figure 4.14: Search Tree with Loops

of this loop are known as t_{29} and t_{30}. We define a vector with pair elements $\overrightarrow{Tloop} = (t_{start}, t_{finish})$. Therefore, loops can be avoided as follow:

$$T^{f}_{x(k+1)}(t_{finish}) = 1 \Rightarrow s_{Rd_{x(k+1)}}(t_{start}) \neq s_{Rd_{xk}}(t_{finish}) \tag{4.50}$$

The equation (4.50) means that when t_{finish} is finished, t_{start} cannot be fired at next step with the same token identifier.

K	Time	Fails	variables	constraints	makespan
Token identification technique in TPNs with safe behavior					
22	1.92 s	2313	70633	75990	182
23	2.29 s	2456	72846	79450	183
Token identification technique in bounded TPNs					
22	0.06 s	6	2333	70633	182
23	0.06 s	6	2477	73824	183

Table 4.7: Reconfigurable Transport System Avoiding Loops of Robots

In table (4.7), results show that solutions can be found easily after avoiding loops of robots. Meanwhile, token identification technique in bounded TPNs is

more efficient than token identification technique in TPNs with safe behavior.

However, this specific method has limits on developing new model and methods. Therefore we develop one general method for avoiding all loops.

4.5.2 General Mechanism to Avoid Loops in TPNs for CP

To avoid loops more generally, we propose to add one new vector – firing priority vector $\overrightarrow{FP_x}$ – to transitions that can contain token identifiers. In $\overrightarrow{FP_x}$, each transition is given a number to represent its priority, and high number will represent high priority. If one transition is fired, the corresponding priority will be set as 0 (transitions with 0 in $\overrightarrow{FP_x}$ will not be fired), then loops can be forbidden. Take figure (4.13) as an example, transitions t_{29}, t_{30} are set as 1 at beginning. When token goes through these transitions at the first time, they will be set as 0, and only t_{33} can be fired. Therefore this loop will be forbidden.

In this thesis, we propose to first set all elements as 1 in $\overrightarrow{FP_x}$ of the reconfigurable transport system, since no priority transitions are needed. Since only loops in the *Pregraph* are needed to be avoided, $\overrightarrow{FP_x}$ is only defined for transitions in *Pregraph*. Such consideration can reduce the number of variables.

But in some cases, tokens need to go through loops a bounded number of times. Thus, the $\overrightarrow{FP_x}$ must be reset to some priority based on *requirements*. Considering the figure (4.6), if a product needs to be operated in f_3, t_2, it can go to M_4 and T_3. However, if T_3 is broken down, product will be reconfigured to go to T_1. This may lead tokens to go through some places several times in the *Pregraph*. Therefore we propose to reset $\overrightarrow{FP_x}$ when tokens with the same token identifier return into *operating sequence*, which can be seen as *requirements*. It means that after the operation of one machine, token can be transported to anywhere in the *Pregraph* again.

These considerations lead to equations below:

$$t_j \in \mathbb{T}^{Pregraph}$$

$$\sum_{t_c \in \mathbb{T}_2} \sigma_{x(k+1)}(t_c) = 0 \wedge \sigma_{x(k+1)j} = 0 \Rightarrow FP_{x(k+1)j} = FP_{xkj} \tag{4.51}$$

$$\sum_{t_c \in \mathbb{T}_2} \sigma_{x(k+1)}(t_c) = 0 \wedge \sigma_{x(k+1)j} = 1 \Rightarrow FP_{x(k+1)j} = 0 \tag{4.52}$$

$$\sum_{t_c \in \mathbb{T}_2} \sigma_{x(k+1)}(t_c) = 1 \Rightarrow FP_{x(k+1)j} = FP_{x0j} \tag{4.53}$$

In equations (4.51, 4.52 and 4.53), $\sum_{t_c \in \mathbb{T}_2} \sigma_{x(k+1)}(t_c)$ can decide if *requirements* are satisfied or not. Then in equations (4.51 and 4.52), the firing priority vector $\overrightarrow{FP_x}$ will be updated based on the new fired transition. If one transition is newly fired, then the corresponding firing priority vector will be set as 0 to forbid it to be fired again until equation (4.53) is satisfied. In equation (4.53) $\overrightarrow{FP_x}$ will be reset to the original value if one transition in $\mathbb{T}_2$ is fired, since it means token has gone back to the *operating sequence* model.

K	Time	Fails	variables	constraints	makespan
Token identifiers in bounded TPNs					
22	0.05 s	1	26695	34896	182
23	0.14 s	1	27867	36480	183
24	0.11 s	1	29039	38064	184

Table 4.8: General method of Avoiding Loops for figure (4.6)

The example in figure (4.6) is resolved and results are given in table (4.8). Comparing with results in table (4.7), the general method for avoiding loops meets less fails. The firing sequences of TPNs with K = 22 for the reconfigurable transport system presented in figure (4.6) is $(t_1t_{18}, 0)$, $(t_{39}, 1)$, $(t_{36}t_{39}, 10)$, $(t_{35}t_{46}, 19)$, $(t_{29}t_{47}, 29)$, $(t_2t_{49}, 38)$, $(t_4t_{55}, 41)$, $(t_6t_{19}, 64)$, $(t_{20}t_{30}, 65)$, $(t_{21}t_{33}, 94)$, $(t_{34}t_{54}, 97)$, $(t_{31}t_{52}, 106)$, $(t_{10}t_{22}, 115)$, $(t_{14}t_{24}, 116)$, $(t_{16}t_{26}, 143)$, $(t_{32}t_{53}, 144)$, $(t_{38}t_{48}, 153)$, $(t_{40}t_{41}, 158)$, $(t_{40}t_{42}, 163)$, $(t_{28}t_{42}, 172)$, $(t_{17}, 181)$, $(\emptyset, 182)$.

In conclusion, the reconfiguration of manufacturing system - reconfigurable transport system - has been addressed with our approaches.

4.6 Conclusion

In this chapter, we have applied our incremental approach to a realistic issue concerning manufacturing systems: the problem of reconfiguration of manufacturing systems. To solve this problem, token confusion issues and loops are addressed with token identification techniques and methods avoiding loops respectively.

In the first section, the main issues of reconfigurable transport system are given based on decomposing it into different TPN models, such as *operating sequence*, *transport system*, etc. Then the token confusion issues are given, and token identification technique of ordinary PN is introduced.

In the second section, we adapt this token identification technique to TPNs. A new vector $\overrightarrow{s_{Rd_k}}$ is defined to represent missing token identifiers. However, it still restricts to TPNs with safe behavior.

In the third section, we propose to associate token identifier vectors $\overrightarrow{s_{Id_x}}$ and $\overrightarrow{s_{Rd_x}}$ to each kind of tokens. These vectors can be seen as new TPNs associate with the original TPN. Such consideration allow applying the token identification technique to bounded TPNs. Even if more variables are needed to express these vectors, it solves token confusion issues with nearly the same time as other token identification techniques.

In the final section, we propose to avoid loops in TPNs by adding a firing priority vector $\overrightarrow{FP_x}$. This vector is associated with each kind of token identifier and is combined with the token identification technique developed for bounded TPNs. Thus, with regard to the token identification technique in TPNs with safe behavior, the firing priority of each token can be well expressed independently. When one transition has been fired, it will be set as 0 to show that this transition cannot be fired again. However, in some cases, tokens need to follow some loops. Therefore, we propose to reset the vector $\overrightarrow{FP_x}$ based on special *requirements* in model.

To conclude, our incremental approach in this thesis has been well applied for a realistic system. It has shown the efficiency for resolving the reachability problems and the ability of solving specific problems in systems.

Chapter 5

Conclusions and Perspectives

Conclusions

In this thesis, we are interested in generating firing sequences in TPNs with Constraint Programming Approach. An *incremental model* has been successfully developed for expressing the whole behavior of TPNs, which has reduced the complexity of *combinatorial explosion*. Then a *constraint programming model* has been developed for implementing the incremental model along with several modeling improvements and search strategies. This process has presented modeling and solving abilities of our CP model. Therefore the application of reconfiguration of manufacturing system has been resolved following our approaches. Even if our incremental model need to be verified with more practical systems (especially large scale systems), results presented in this thesis has brought several useful improvements. The work has been presented as follows:

- The first chapter has introduced the motivation, problems and challenges, several general methods for solving these problems, and the contributions and organizations of this thesis.

  Our motivation is to generate firing sequences in TPNs, which can be divided into two steps:

  - Solve the ***reachability problem*** of TPNs.
  - Give timed ***firing sequences*** that leading to the desired marking.

The main problem to generate firing sequences in TPNs is the *combinatorial explosion* of states, for which many methods have been proposed. The figure (5.1) is a search tree.

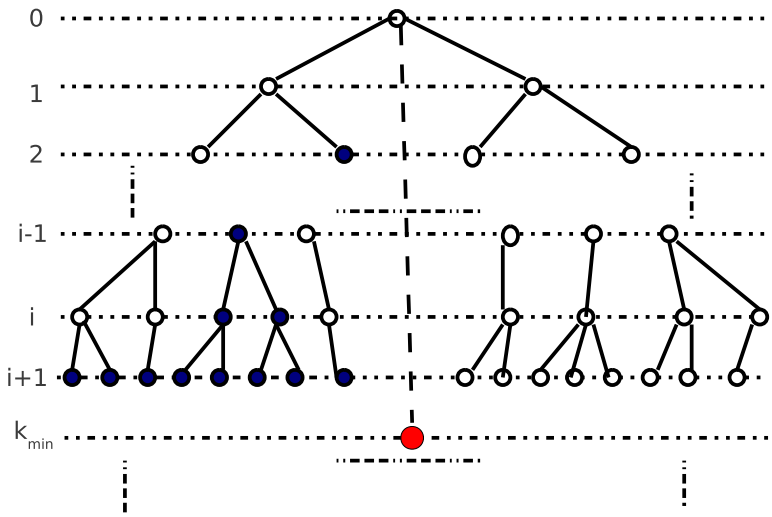


Figure 5.1: A Search Tree

Firstly, we propose to follow the idea of BMC to explore this search tree *incrementally*, since the whole underlying reachability graph is not needed to be explored. It means that solver will explore the search tree under a search depth k and continue to search deeply by increasing this depth k, if there is no solution found or the upper bound is reached.

Secondly, we propose to use the *logical abstraction technique*, which uses timed steps of length k to implicitly represent the behavior of TPNs within k steps.

Finally, we propose to use several developed search strategies to lead solver to explore the search tree more efficiently. In the best situation, if k_{min} is first given, the solution can be found with meeting no deadends.

- The second chapter has developed the fundamental contribution of this thesis: incremental model for TPNs. The formal definition of TPNs and their terminology have been introduced based on [CC88], where *Controlled Execution* are defined for presenting the instantaneous state and reachability

problem of TPNs.

The formal definition of timed step and timed step *firings* are defined to express the incremental model. Then the equivalence between timed step firing and controlled executions is proved to show the correctness and completeness of our incremental approach. Based on this proof, a collary is given to express the reachability problem of TPNs by timed step firings.

At the end, several incremental search algorithms are introduced to improve the process of searching. We have applied the incremental search that will restart the search from the root node, since timed steps and search strategies are more suitable to this method.

- The third chapter has translated the incremental model for constraint programming and developed several modeling improvements and search strategies to enforce the efficiency. Then the problem of generating firing sequences of TPNs has been translated as searching for solutions in the *search space* of a CP model. To find solutions more quickly, modeling improvements (presented in subsection (3.2.2)) has reduced the search space and search strategies (presented in (3.3)) has reduced the chance of meeting fails in the *search tree*. The time needed for searching for solutions varies dramatically among different search strategies. Labeling variables σ_k, s_{mk}, Δ_{k-1} and s_{rk} with a *step-by-step* manner have been the most efficient search strategies, since this developed search strategy is developed just as the incremental approach to express the behavior of TPNs.

- The fourth chapter has applied the method presented in chapters (2) and (3) to the reconfiguration of manufacturing systems.

  We express the reconfiguration process of an FMS between two configurations as a reachability problems. Reconfiguration actions can be addressed by firing sequences. Before generating firing sequences, token confusion issues and loops have been addressed with token identification techniques and methods avoiding loop respectively.

  Token identification technique for bounded TPNs has been derived from our original work [HBYT12a], and has been developed more general and

extendable. We have proposed a token identifier layer in TPNs for expressing the behavior of each kind of tokens. Such consideration can resolve the token confusion issues by structural modeling.

Then, a general method for avoiding loops has been proposed by adding a firing priority vector. This vector records the firing priority of transitions and forbids transitions to be fired under some requirement.

At last, benchmarks are given to present the efficiency of our approaches.

Perspectives

In the future, we propose to follow interesting directions raised by this thesis:

- In the section (2.3), we have mentioned several search algorithms for our incremental model, as shown in the figure (5.2). However, we have mainly used the fixed depth TPN reachability problem in the following chapters. This algorithm will explore the search tree from the root node each time. This algorithm is a semi-decision algorithm. Indeed, in the context of unbounded TPNs, the number of firing dates K is set arbitrarily, as we do not know any information on the number of steps needed to find a possible solution. Thus, if no solution is obtained using this value K, one cannot conclude on the reachability property. To the contrary, when dealing with bounded TPNs, it is possible to set K to the value of the *"sequential depth"* of the net, a parameter defined in [BHY04] and which guarantee the complete exploration of the reachability graph. Using this parameter as search depth, it is always possible to conclude when the algorithm stops. A promising research track could be to define efficient procedures to evaluate the value of this parameter.

- Also in the section (2.3), we have used the incremental search algorithm b in the figure (5.2). Since the sequential depth K is hardly to be solved as the reachability problems, it is simply to see that algorithms c and d are more efficient, since they will use efforts at last search depth to improve the search at next depth. They avoid to search for lower depth each time. In fact, for algorithm d, the jump search can also use the benefits of timed steps, timed step firings and search strategies.

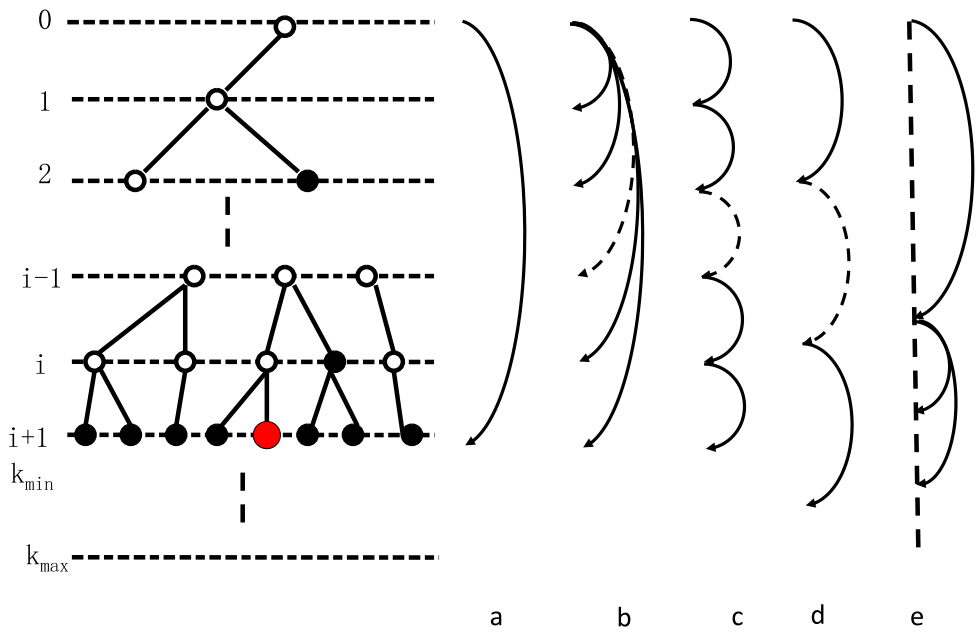


Figure 5.2: Incremental Search Algorithms

- The CP model could be more completely studied. In the *Modeling* part, more work should be studied to know the difference between CP model and IP model. It means that the working mechanism of conditional constraints and the linearization technique should be studied. Meanwhile, more constraints (for example equivalence class) could be applied for reducing search space. In the *Solving* part, more generic search strategies could be developed based on the TPNs model. To better control the solving process, more works could be done following [RBW06], for example filtering algorithms, nogood recording, etc.

- The token identification technique for bounded TPNs could be extended to a simple colored PNs. In fact, the ID can be seen as one kind of color in colored PNs. More colors could be associated with each token identifier, which could improve the expressivity of our model.

- The general method for avoiding loops can be extended for more realistic systems. The firing priority vector $\overrightarrow{FP_x}$ can be initialized with different numbers, which indicate different priority of different transitions. For ex-

ample, specific tokens can be processed first than tokens with lower priority, if they can both be used.

- The diagnostic / diagnosis problem could be applied with our approach, since the fault behavior and unobservable states could be seen as reachability problem or some actions in the firing sequences. Meanwhile, the key problem in this area is also the combinatorial explosion problem. Therefore, our work can be the first step to combine these areas, which is the motivation of our team.

List of Figures

List of Tables

LIST OF TABLES

Appendix A

Proposition .1 (Equivalence between CE and timed step firing sequences). *Let (R, d) be a TPN with its initial state $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ given at date $v_0 = 0$. Let $s_f = \left(\overrightarrow{s_{m_f}}, \overrightarrow{0_{\mathrm{N}}}\right) \in \mathscr{S}(R, d)$ be a state.*

There exists a feasible controlled execution allowing to reach s_f at date $v_{\max}$ from $s_0 \Leftrightarrow$

$$\begin{array}{l} \exists \mathrm{K} \in \mathbb{N}, \\ \exists v_1, v_2, \ldots, v_{\mathrm{K}} \in \mathbb{N} \\ \exists s_1, s_2, \ldots, s_{\mathrm{K}} \in \mathscr{S}(R, d). \\ \exists \psi_1 = (\sigma_1, v_1), \psi_2, \ldots, \psi_{\mathrm{K}} = (\sigma_{\mathrm{K}}, v_{\mathrm{K}}), \\ \qquad\qquad \psi_{\max} = (\overrightarrow{0_{\mathrm{N}}}, v_{\max}) \in \mathbb{T}^*_{TPN} \end{array} \quad s.t. : \left\{ \begin{array}{r} \forall k \in [\![1, \mathrm{K}]\!], s_{k-1}[\psi_k\rangle s_k \\ s_{\mathrm{K}}[\psi_{\max}\rangle s_f \end{array} \right. \tag{1}$$

Proof. ($\Rightarrow$) Let $(u_k^t)_{t \in \mathbb{T}, k \in [\![1, K_t]\!]}$ be the CE such that $s_{v_{\max}} = \left(\overrightarrow{s_m(v_{\max})}, \overrightarrow{s_r(v_{\max})}\right) = s_f$. We buid a step sequence from (u_k^t) as follows. Let $v_1, v_2, \ldots, v_{\mathrm{K}} \in [\![0, v_{\max}]\!]$ be the list of the firing dates of the controlled execution, sorted in a growing order. Formally we have: $\bigcup_{t \in \mathbb{T}, k \in [\![1, K_t]\!]} u_k^t = \{v_1, v_2, \ldots, v_{\mathrm{K}}\} = \mathbb{V}$. At each date $v_k \in \mathbb{V}$, we build a step $\sigma_k = \sum_{j \in [\![1, \mathrm{N}]\!]} \alpha_j^k \cdot t_j$ such that $\sigma_k = \{t \in \mathbb{T} \mid \exists r \in [\![1, \mathrm{K}_t]\!],\ u_r^t = v_k\}$.

From equation (2.1), we have obviously $\forall j \in [\![1, \mathrm{N}]\!], \alpha_j^k \in \{0, 1\}$, thus the *step* (σ_k) can be used to build timed steps $\psi_k = (\sigma_k, v_k)$ satisfying the *non-reentrance condition* of definition (2.9).

We will show that there exists K states $s_1, s_2, \ldots, s_{\mathrm{K}} \in \mathscr{S}(R, d)$ such that $\forall k \in [\![1, \mathrm{K}]\!], s_{k-1}[\psi_k\rangle s_k$, and that these states are precisely the *instantaneous states* reached by the CE at the respective dates $v_1, v_2, \ldots, v_{\mathrm{K}}$. For each $k \in [\![1, \mathrm{K}]\!]$, let $s_{v_k} = \left(\overrightarrow{s_m(v_k)}, \overrightarrow{s_r(v_k)}\right)$ be the instantaneous state reached by the considered CE at date v_k. We use an induction on r to prove the following property:

$$\boxed{(I_r) : \text{For each } k \in [\![1, r]\!],\ s_{v_{k-1}}[\psi_k\rangle s_{v_k}}$$

$\boxed{r = 1}$ The timed step $\psi_1 = (\sigma_1, v_1)$ is associated to the first firing date v_1 of the controlled execution. Let Δ_{v_0} be the delay between $v_0 = 0$ and v_1. Since $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$, we have obviously $T_1^f = \{t \in \mathbb{T}, \overrightarrow{s_{r_0}}(t) = 0\} = \varnothing$, $T_1^s = \{t \in \mathbb{T}, \overrightarrow{s_{r_0}}(t) - \Delta_{v_0} \leqslant 0\} = \varnothing$. Then $\forall t \in \sigma_1, \overrightarrow{s_{r_0}}(t) = 0 \leqslant \Delta_{v_0} = v_1$, which satisfy the equation (2.7).

Let us consider the instantaneous state reached by the CE at date v_1. Since the CE is feasible, we have from equation (2.4): $\overrightarrow{s_m(v_1)} \geqslant \overrightarrow{0_{\mathrm{M}}}$. Thus, equation (2.2) gives us: $\overrightarrow{s_{m_0}} + C^+ \cdot \overrightarrow{F(v_1)} - C^- \cdot \overrightarrow{N(v_1)} \geqslant \overrightarrow{0_{\mathrm{M}}}$. From the definition of $\overrightarrow{F(v_1)}$ and $\overrightarrow{N(v_1)}$, we get: $\overrightarrow{F(v_1)} = \overrightarrow{0_{\mathrm{N}}}$ and $\overrightarrow{N(v_1)} = \overrightarrow{\sigma_1}$. We obtain finally: $\overrightarrow{S_{m_0}} - C^- \cdot \overrightarrow{\sigma_1} \geqslant \overrightarrow{0_{\mathrm{M}}}$, which corresponds to equation (2.8). Since no transition were active initially, the timed step $\psi_1 = (\sigma_1, v_1)$ is **fireable** at date v_1.

Then, we will show that both controlled executions and step sequences formulations give the same reached state at date v_1.

- Since $\overrightarrow{F(v_1)} = \overrightarrow{0_{\mathrm{N}}}$, we have obviously: $\overrightarrow{s_m(v_1)} = \overrightarrow{s_{m_0}} - C^- \cdot \overrightarrow{\sigma_1} = \overrightarrow{s_{m_1}}$.

- Since v_1 is defined as the first firing date of the CE, all transitions that are not fired exactly at that date (i.e. that do not belong to σ_1) start strictly after v_1: $\forall t \notin \sigma_1, \forall k \in [\![1, \mathrm{K}_t]\!], u_k^t > v_1$. At the contrary, $\forall t \in \sigma_1, u_1^t = v_1$. Thus, equation (2.3) becomes:

$$\forall t \in \mathbb{T}, \left.\overrightarrow{s_r(v_1)}\right|_t = \begin{cases} v_1 + d(t) - v_1 = d(t) & \text{if } t \in \sigma_1 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

  In the other hand, since $\forall t \in \mathbb{T}, \overrightarrow{s_{r_0}}(t) = 0 \leqslant \Delta_{v_1} = v_1$, equation (2.10) becomes:

$$\forall t \in \mathbb{T}, \overrightarrow{s_{r_1}}(t) = \begin{cases} d(t) & \text{if } t \in \sigma_1 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Finally, equations (2) and (3) denote the same residual durations vector, which concludes the first step of the induction.

$\boxed{r \rightsquigarrow r+1}$ Let us assume that I_r is true for $r = g < \mathrm{K}$. From the induction hypothesis, we know that $\forall k \in [\![1, g]\!], s_{v_{k-1}}[\psi_k\rangle s_{v_k}$. We will show that ψ_{g+1} is a timed step fireable from s_{v_g} and that the reached state $s_{v_{g+1}}$ is exactly the instantaneous state reached by the CE at date v_{g+1}. Let Δ_{v_g} be the delay between v_g and v_{g+1}.

We first show that any previous activation of a transition fired in ψ_{g+1} is already finished at date v_{g+1}. By reductio ad absurdum: let us assume that $\exists t \in \sigma_{g+1}$ s.t. $\overrightarrow{s_r(v_g)}\Big|_t > \Delta_{v_g}$. From equation (2.3), we get: $\exists k \in [\![1, \mathrm{K}_t]\!]$ s.t. $u_k^t + d(t) - v_g = \overrightarrow{S_r(v_g)}\Big|_t$. Thus, $u_k^t + d(t) - v_g > v_{g+1} - v_g$, i.e. $u_k^t + d(t) > v_{g+1}$. Since $t \in \sigma_{g+1}$, t is fired at date v_{g+1} in the CE, thus $\exists k' > k \in [\![1, \mathrm{K}_t]\!]$ s.t. $v_{g+1} = u_{k'}^t$. Then we get: $\exists k < k' \in [\![1, \mathrm{K}_t]\!]$ s.t. $u_k^t + d(t) > u_{k'}^t$, which *contradicts* the equation (2.1) given in the definition of a CE. Finally, equation (2.7) is satisfied by ψ_{g+1}. To show that ψ_{g+1} is fireable, we must now consider equation (2.8).

Since the CE is feasible, we have: $\overrightarrow{s_m(v_{g+1})} \geqslant \overrightarrow{0}_{\mathrm{M}}$. Using the definition of instantaneous firings (2.2) and characteristic vectors of CE (2.4), we get:

$$
\begin{aligned}
& \overrightarrow{s_m(v_{g+1})} = \overrightarrow{s_{m_0}} + C^+ \cdot \overrightarrow{F(v_{g+1})} - C^- \cdot \overrightarrow{N(v_{g+1})} \geqslant \overrightarrow{0}_{\mathrm{M}} \\
\Leftrightarrow\ & \overrightarrow{s_m(v_{g+1})} = \overrightarrow{s_{m_0}} + C^+ \cdot \left(\overrightarrow{F(v_g)} + \sum_{t \in \mathbb{T}} card\left(\left\{ u_k^t, k \in [\![1, \mathrm{K}_t]\!] \;\middle|\; \begin{array}{l} u_k^t + d(t) \leqslant v_{g+1} \\ \wedge \quad u_k^t + d(t) > v_g \end{array} \right\} \right) \cdot \overrightarrow{e_t} \right) \\
& \qquad - C^- \cdot \left(\overrightarrow{N(v_g)} + \sum_{t \in \mathbb{T}} card\left(\left\{ u_k^t, k \in [\![1, \mathrm{K}_t]\!] \;\middle|\; \begin{array}{l} u_k^t \leqslant v_{g+1} \\ \wedge \quad u_k^t > v_g \end{array} \right\} \right) \cdot \overrightarrow{e_t} \right) \geqslant \overrightarrow{0}_{\mathrm{M}} \\
\Leftrightarrow\ & \overrightarrow{s_m(v_{g+1})} = \overrightarrow{s_m(v_g)} + C^+ \cdot \sum_{t \in \mathbb{T}} card\left(\left\{ u_k^t, k \in [\![1, \mathrm{K}_t]\!] \;\middle|\; \begin{array}{l} u_k^t + d(t) \leqslant v_{g+1} \\ \wedge \quad u_k^t + d(t) > v_g \end{array} \right\} \right) \cdot \overrightarrow{e_t} \\
& \qquad - C^- \cdot \sum_{t \in \mathbb{T}} card\left(\left\{ u_k^t, k \in [\![1, \mathrm{K}_t]\!] \;\middle|\; \begin{array}{l} u_k^t \leqslant v_{g+1} \\ \wedge \quad u_k^t > v_g \end{array} \right\} \right) \cdot \overrightarrow{e_t} \geqslant \overrightarrow{0}_{\mathrm{M}}
\end{aligned}
\tag{4}
$$

Let us consider the vectors $\overrightarrow{f} = \sum_{t \in \mathbb{T}} card\left(\left\{ u_k^t, k \in [\![1, \mathrm{K}_t]\!] \;\middle|\; \begin{array}{l} u_k^t + d(t) \leqslant v_{g+1} \\ \wedge \quad u_k^t + d(t) > v_g \end{array} \right\} \right) \cdot \overrightarrow{e_t}$ and $\overrightarrow{n} = \sum_{t \in \mathbb{T}} card\left(\left\{ u_k^t, k \in [\![1, \mathrm{K}_t]\!] \;\middle|\; \begin{array}{l} u_k^t \leqslant v_{g+1} \\ \wedge \quad u_k^t > v_g \end{array} \right\} \right) \cdot \overrightarrow{e_t}$ used above. From

the definition of the firing dates $(v_k)_{[\![1,\mathrm{K}]\!]}$, we have obviously: $\overrightarrow{n} = \overrightarrow{\sigma_{g+1}}$. We will show that $\overrightarrow{f} = \sum_{t \in T^f_{g+1}} \overrightarrow{e_t}$.

For each $t \in \mathbb{T}$, there can be only one $k \in [\![1, \mathrm{K}_t]\!]$ such that $\begin{cases} & u^t_k + d(t) \leqslant v_{g+1} \\ \wedge & u^t_k + d(t) > v_g \end{cases}$ (otherwise, since there cannot be two firings of the same transition before date v_g, and $d(t) > 0$, there must be a firing strictly between v_g and v_{g+1}, which contradicts the definition of $(v_k)_{[\![1,\mathrm{K}]\!]}$).

Given $\mathscr{A} = \left\{ t \in \mathbb{T}, \exists k \in [\![1, \mathrm{K}_t]\!], \begin{cases} & u^t_k + d(t) \leqslant v_{g+1} \\ \wedge & u^t_k + d(t) > v_g \end{cases} \right\}$, one can thus rewrite $\overrightarrow{f} = \sum_{\mathscr{A}} \overrightarrow{e_t}$. To conclude, it is now sufficient to verify that $T^f_{g+1} = \mathscr{A}$. We proceed by a mutual inclusion.

Let $t \in T^f_{g+1}$. From equation (2.5), we get: $\exists i \in [\![1, \mathrm{K}]\!]$ s.t. $\sigma_i(t) = 1, v_g < v_i + d(t) \leqslant v_{g+1}$. For $\bigcup_{t \in \mathbb{T}, k \in [\![1, K_t]\!]} u^t_k = \{v_1, v_2, \ldots, v_{\mathrm{K}}\} = \mathbb{V}$. we can always find only one controlled execution firing date $u^t_k = v_i$ s.t. $k \in [\![1, \mathrm{K}_t]\!]$ s.t. $v_g < u^t_k + d(t) \leqslant v_{g+1} : t \in \mathscr{A}$.

Let $t \in \mathscr{A}$. since $\exists k \in [\![1, \mathrm{K}_t]\!]$ s.t. $v_g < u^t_k + d(t)$, $u^t_k + d(t) \leqslant v_{g+1}$. Similarly, we can always find one firing date $v_i = u^t_k$, where $i \in [\![1, g]\!]$ s.t. $\sigma_i(t) = 1, v_g < v_i + d(t) \leqslant v_{g+1}$. And this firing date $v_i \leqslant v_g$ must be the latest fired $\max_{p \in [\![1,g]\!]} \{v_p : \sigma_p(t) = 1\}$. For if there is another $v_j > v_i, j \in [\![1, g]\!]$, for $v_g < v_i + d(t) \leqslant v_{g+1}$ and transition cannot reentrance leading to $v_j > v_g$. So $t \in T^f_{g+1}$.

We have shown that $\overrightarrow{f} = \sum_{t \in T^f_{g+1}} \overrightarrow{e_t}$ and $\overrightarrow{n} = \overrightarrow{\sigma_{g+1}}$. Finally equation (4) becomes:

$$\overrightarrow{s_m(v_{g+1})} = \overrightarrow{s_m(v_g)} + C^+ \cdot \sum_{t \in T^f_{g+1}} \overrightarrow{e_t} - C^- \cdot \overrightarrow{\sigma_{g+1}} \geqslant \overrightarrow{0_{\mathrm{M}}} \tag{5}$$

Since the equation above corresponds to equation (2.8) the timed step ψ_{g+1} is fireable from s_g.

Moreover, equation (5) shows the marking reached by ψ_{g+1} is equal to $\overrightarrow{s_m(v_{g+1})}$. To conclude the induction, we will prove that the residual duration vector $\overrightarrow{s_{r_{g+1}}}$ reached by ψ_{g+1} at date v_{g+1} is exactly $\overrightarrow{s_r(v_{g+1})}$. For $\overrightarrow{s_{r_g}} = \overrightarrow{s_r(v_g)}$ at date v_g and

the definition (2.11), we have:

$$\forall t \in \mathbb{T}, \overrightarrow{s_{r_{g+1}}}(t) = \begin{cases} d(t) & \text{if } t \in \sigma_{g+1} \\ \overrightarrow{s_r(v_g)}\Big|_t - \Delta_{v_g} & \text{if } t \in T^s_{g+1} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Let $t \in T^s_{g+1}$ as shown in equation (2.6). So this fired transition t before date v_{g+1}, satisfy $v_{g+1} < \max_{i \in [\![1,g]\!]} \{v_i : \sigma_i(t) = 1\} + d(t)$. Let us assume that $\exists k \in [\![1, \mathrm{K}_t]\!]$ s.t. $v_{g+1} \in]u^t_k, u^t_k + d(t)[\![, u^t_k = v_i$. Then we have: $\begin{cases} u^t_k < v_{g+1} \\ v_{g+1} < u^t_k + d(t) \end{cases}$

$$\Rightarrow \begin{cases} u^t_k \leqslant v_g \wedge v_g < u^t_k + d(t) \\ v_{g+1} - v_g < u^t_k + d(t) - v_g \end{cases} \Rightarrow \begin{cases} v_g \in [\![u^t_k, u^t_k + d(t)[\![\\ \Delta_{v_g} < u^t_k + d(t) - v_g \end{cases}$$

$$\overset{\text{eq (2.3)}}{\Rightarrow} \begin{cases} \overrightarrow{s_r(v_g)}\Big|_t = u^t_k + d(t) - v_g \\ \Delta_{v_g} < \overrightarrow{s_r(v_g)}\Big|_t \end{cases} \Rightarrow \begin{cases} \overrightarrow{s_r(v_g)}\Big|_t - \Delta_{v_g} = u^t_k + d(t) - v_{g+1} \\ \Delta_{v_g} < \overrightarrow{s_r(v_g)}\Big|_t \end{cases}.$$

Then:

$$(6) \Rightarrow \forall t \in \mathbb{T}, \overrightarrow{s_{r_{g+1}}}(t) = \begin{cases} d(t) & \text{if } \exists k \in [\![1, \mathrm{K}_t]\!] \text{ s.t. } u^t_k = v_{g+1} \\ u^t_k + d(t) - v_{g+1} & \text{if } \exists k \in [\![1, \mathrm{K}_t]\!] \text{ s.t. } v_{g+1} \in]u^t_k, u^t_k + d(t)[\![\\ 0 & \text{otherwise} \end{cases}$$

$$\Rightarrow \forall t \in \mathbb{T}, \overrightarrow{s_{r_{g+1}}}(t) = \begin{cases} u^t_k + d(t) - v_{g+1} & \text{if } \exists k \in [\![1, \mathrm{K}_t]\!] \text{ s.t. } v_{g+1} \in [\![u^t_k, u^t_k + d(t)[\![\\ 0 & \text{otherwise} \end{cases}$$

$\overset{\text{def 2.5}}{\Rightarrow} \overrightarrow{s_{r_{g+1}}} = \overrightarrow{s_r(v_{g+1})}$, which concludes the induction process.

We use the property I_r with the whole set of timed steps to prove that $\Psi = (\psi_1, \psi_2, \ldots, \psi_{\mathrm{K}})$ is a timed step sequence that leads from s_0 to the instantaneous state s_{K} reached at the last firing date of the CE. Since the final state is given at date $v_{\max}$, we must prove that there exists a timed step $\psi_{\max}$ allowing to reach s_f from s_{K} at date $v_{\max}$. Obviously, we set $\psi_{\max} = (\overrightarrow{0_{\mathrm{N}}}, v_{\max})$. Since it is empty, this timed step complies with non-reentrance condition of definition (2.9) and with fireability conditions (2.7) and (2.8). We could use the same argumentation as before to show that the state reached at date $v_{\max}$ by the timed step sequence is

identical to the instantaneous state reached by the CE at the same date.

($\Leftarrow$)

Let $\Psi = \Big(\psi_1 = (\sigma_1, v_1),\ \ldots,\ \psi_{\mathrm{K}} = (\sigma_{\mathrm{K}}, v_{\mathrm{K}}),\ \psi_{\max} = (\overrightarrow{0_{\mathrm{N}}}, v_{\max})\Big)$ be a fireable timed step sequence leading from s_0 to s_f. Let $(s_k = (\overrightarrow{s_{m_k}}, \overrightarrow{s_{r_k}}))_{k\in[\![1,\mathrm{K}]\!]}$ be the states reached by the timed step sequence such that $s_0[\psi_1\rangle s_1 = [\ldots\rangle s_{\mathrm{K}-1}[\psi_{\mathrm{K}}\rangle s_{\mathrm{K}}[\psi_{\max}\rangle s_f$. Let $\mathbb{V} = \{v_1, v_2, \ldots, v_{\mathrm{K}}, v_{\max}\}$. Let $(u_k^t)_{t\in\mathbb{T},k\in[\![1,\mathrm{K}_t]\!]}$ be a controlled execution defined by:

- $\forall t \in \mathbb{T}, \mathrm{K}_t = card(\{k \in [\![1,\mathrm{K}]\!], t \in \sigma_k\})$;
- $\forall t \in \mathbb{T}$, $(u_k^t) = v_1^t, v_2^t, \ldots, v_{\mathrm{K}_t}^t \in \mathbb{V}$ is the increasing sequence of the firing dates of t in the timed step sequence: $\forall r \in [\![1,\mathrm{K}_t]\!], \exists k \in [\![1,\mathrm{K}]\!], v_k = v_r^t$ and $t \in \sigma_k$.

We will show that (u_k^t) is a feasible controlled execution leading to s_f from s_0 at date $v_{\max}$.

First, we show that (u_k^t) is a well defined controlled execution that complies with the non-reentrance condition (2.1). By reductio ad absurdum: suppose there exists $t \in \mathbb{T}, k \in [\![1,\mathrm{K}_t]\!]$ s.t. $u_k^t + d(t) > u_{k+1}^t$. Let $\psi_i, \psi_j \in \Psi$ be the timed steps associated with the firing dates u_k^t and u_{k+1}^t. We have: $s_{i-1}[\psi_i\rangle s_i[\psi_{i+1}\rangle s_{i+1}[\ldots\rangle s_{j-1}[\psi_j\rangle s_j$ and $v_i + d(t) > v_j$.

Since ψ_i is fired at date $u_k^t = v_i$, we have from (2.10): $\overrightarrow{s_r(v_i)}\Big|_t = d(t)$. We show by a simple induction on $r \leqslant (j-i)$ the property:

$$(I_r) : \overrightarrow{s_r(v_{i+r})}\Big|_t = d(t) - (v_{i+r} - v_i)$$

$\boxed{r=1}$ $\overrightarrow{s_r}(v_i)\big|_t - (v_{i+1} - v_i) \overset{\overrightarrow{s_r}(v_i)\big|_t = d(t)}{=} d(t) - v_{i+1} + v_i \overset{v_{i+1}\leqslant v_j}{\geqslant} d(t) - v_j + v_i \overset{v_i + d(t) > v_j}{>} 0$. Thus, we get from (2.10): $\overrightarrow{s_r(v_{i+1})}\Big|_t = \overrightarrow{s_r(v_i)}\Big|_t - (v_{i+1} - v_i) = d(t) - (v_{i+1} - v_i)$.

$\boxed{r \rightsquigarrow r+1}$ Let us assume that I_r is true for $r < j-i$. We have: $v_{i+r+1} < v_{j+1}$, i.e. $v_{i+r+1} \leqslant v_j$. Then: $\overrightarrow{s_r(v_{i+r})}\Big|_t - (v_{i+r+1} - v_{i+r}) \overset{\overrightarrow{s_r(v_{i+r})}\big|_t = d(t) - (v_{i+r} - v_i)}{=} d(t) - (v_{i+r} - v_i) - (v_{i+r+1} - v_{i+r}) = d(t) - (v_{i+r+1} - v_i) \overset{v_{i+r+1}\leqslant v_j}{\geqslant} d(t) - v_j + v_i \overset{v_i + d(t) > v_j}{>} 0$.

Thus, we get from (2.10): $\overrightarrow{s_r(v_{i+r+1})}\Big|_t = \overrightarrow{s_r(v_{i+r})}\Big|_t - (v_{i+r+1} - v_{i+r}) = d(t) - (v_{i+r+1} - v_i)$, which concludes the induction.

We use the property I_r with $r = (j-i-1)$. Then we have: $\overrightarrow{s_r(v_{j-1})}\Big|_t = d(t) - (v_{j-1} - v_i)$. since ψ_j is fireable from s_{j-1}, we must have from (2.7): $\overrightarrow{s_r(v_{j-1})}\Big|_t \leqslant v_j - v_{j-1}$ i.e. $d(t) - (v_{j-1} - v_i) \leqslant v_j - v_{j-1} \Leftrightarrow d(t) + v_i \leqslant v_j$, which contradicts the initial hypothesis: the considered controlled execution respects the *non-reentrance* condition (2.1).

For each $v \in [\![0, v_{\max}]\!]$, we denote by $s'_k = (\overrightarrow{s'_m(v_k)}, \overrightarrow{s'_r(v_k)})$ the instantaneous marking reached by the considered CE at date v. Will we show that the controlled execution is feasible, i.e. that the instantaneous marking reached at any date compatible with the CE is made of positive components. Formally: $\forall v \in [\![0, v_{\max}]\!]$, $\overrightarrow{s'_m(v)} \geqslant \overrightarrow{0_{\text{M}}}$.

We use an induction on r to prove the following property:

$(I_r) : \forall k \in [\![1, r]\!]$, the CE is feasible within $[\![0, v_k]\!]$ and $s_k = s'_k$

$\boxed{r = 1}$ Obviously, the marking under a CE does not change until this first transition is fired, thus we have: $\forall v \in [\![0, v_1[\![$, $\overrightarrow{s'_m(v)} = s_{m_0} \geqslant \overrightarrow{0_{\text{M}}}$. The instantaneous state reached at date v_1 is given by definition (2.5). We have as shown above (step 1 of the first induction): $\overrightarrow{s'_m(v_1)} = \overrightarrow{s_{m_0}} - C^- \cdot \overrightarrow{\sigma_1}$ and $\forall t \in \mathbb{T}$, $\overrightarrow{s'_r(v_1)}\Big|_t = d(t)$. The same state is reached after the timed step ψ_1 at date v_1, which allows to verify that $\overrightarrow{s_m(v_1)} \geqslant 0$ since ψ_1 is fireable. Thus, the CE is fireable within $[\![0, v_1]\!]$.

$\boxed{r \rightsquigarrow r+1}$ Let us assume that I_r is true for $r = g < \text{K}$: the CE is feasible within $[\![1, v_g]\!]$ and $s_g = s'_g$. We will show that the CE is also feasible within $]\!]v_g + 1, v_{g+1}]\!]$. This is obviously true within $]\!]v_g + 1, v_{g+1}[\![$. Indeed: $\forall v \in]\!]v_g + 1, v_{g+1}[\![$, $\overrightarrow{N(v)} = \overrightarrow{N(v_g)}$ and $\overrightarrow{F(v)} \geqslant \overrightarrow{F(v_g)}$, thus $\forall v \in]\!]v_g + 1, v_{g+1}[\![$, $\overrightarrow{s'_m(v)} = \overrightarrow{s_{m_0}} + C^+ \cdot \overrightarrow{F(v)} - C^- \cdot \overrightarrow{N(v)} \geqslant \overrightarrow{s_{m_0}} + C^+ \cdot \overrightarrow{F(v_g)} - C^- \cdot \overrightarrow{N(v_g)} \geqslant \overrightarrow{s'_m(v_g)}$.

Using the same reasoning as above to obtain equation (5), we have moreover:

$$\overrightarrow{s'_m(v_{g+1})} = \overrightarrow{s_m(v_g)} + C^+ \cdot \sum_{t \in T^f_{g+1}} \overrightarrow{s_t} - C^- \cdot \overrightarrow{\sigma_{g+1}} \tag{7}$$

Since the step σ_{g+1} is fireable, and the above equation corresponds to its fireability

condition, we have: $\overrightarrow{s'_m(v_{g+1})} \geqslant \overrightarrow{0_{\mathrm{M}}}$. Thus, the CE is fireable within $]\!]v_g+1, v_{g+1}]\!]$. To conclude, we need to show that $\overrightarrow{s'_r(v_{g+1})} = \overrightarrow{s_r(v_{g+1})}$. Here again, the previous reasoning allows to obtain the desired result, and to conclude the induction.

We use the property I_r with $r = \mathrm{K}-1$ to show that the CE is feasible and allows to reach s_{K} at date v_{K}. Now, let us consider the state $s_{v_{\max}}$ reached by the CE at date $v_{\max}$. since there is not any firing after v_{K}, we have: $\forall v \in]\!]v_{\mathrm{K}}, v_{\max}]\!]$, $\overrightarrow{s_m(v)} \geqslant \overrightarrow{s_m(v_{\mathrm{K}})} \geqslant \overrightarrow{0_{\mathrm{M}}}$, thus the CE is feasible until $v_{\max}$. We can use the same reasoning as before to show that the state reached at date $v_{\max}$ by the CE is identical to the one reached by the timed step sequence, i.e. $s_{v_{\max}} = (\overrightarrow{s_m(v_{\max})}, \overrightarrow{0_{\mathrm{N}}})$.

Then, equation (2.3) gives:

$$\begin{aligned}
&\forall t \in \mathbb{T}, \nexists k \in [\![1, \mathrm{K}_t]\!] \ s.t. \ v_{\max} \in [\![u_k^t, u_k^t + d(t)[\![\\
\Rightarrow &\forall t \in \mathbb{T}, \forall k \in [\![1, \mathrm{K}_t]\!], \ v_{\max} \geqslant u_k^t + d(t) \\
\Rightarrow &v_{\max} = \max_{t \in \mathbb{T}}(u_{\mathrm{K}_t}^t + d(t))
\end{aligned}$$

which corresponds well to the definition of the date $v_{\max}$ above.

□

Résumé étendu en Français

Les réseaux de Petri Temporisés (RdPT) sont utilisés pour modéliser une large classe de systèmes dynamiques discrets. La plupart des problèmes considérés peuvent être modélisés comme des problèmes d'accessibilité, et sont généralement résolus à l'aide d'approches d'exploration du graphe de comportement du réseau de Petri (RdP). Dans cette thèse, nous présentons une modélisation incrémentale du comportement des réseaux de Petri temporisés, permettant de rechercher des séquences de franchissement solution du problème d'accessibilité à l'aide de la programmation par contraintes. Notre approche ne nécessite pas la construction a priosi du graphe d'accessibilité du réseau considéré. Elle permet une recherche efficace de séquences de franchissement solutions des problèmes modélisés. Notre approche est aussi générale que possible puisqu'elle n'impose pas de structure particulière pour le Réseau de Petri, ni de sémantique de franchissment au plus tôt.

Dans cette thèse, il y 5 chapitre.

- Chapitre 1, nous proposons les problèmes et les contextes dans cette thèse.

- Chapitre 2, nous donnons la définition des RdPT et du problème d'accessibilité. Ensuite, nous montrons l'exactitude et la faisabilité de nos méthodes – *l'approache incrémentale*– en prouvant l'équivalence entre l'exécution contrôle et la séquence de steps temporisés.

- Chapitre 3, nous construisons notre modèle incrémental l'utilisant la programmation par contraintes. Le modèle est ensuite amélioré en ajoutant des contraintes supplémentaires. Ensuite, les stratégies de recherche sont appliquées pour améliorer l'efficacité de la recherche.

- Chapitre 4, nous appliquons notre approche incrémentale à la reconfiguration des systèmes de production manufacturière et nous montrons son efficacité. Dans ce chapitre, nous développons des techniques d'identification du jeton permettant d'éviter la confusion de jetons faussant la résolution. D'autre part, nous proposons des solutions à la gestion des boucles du modèles pouvant conduire à des itérations infinis lors de la résolution.

  Les techniques de jeton d'identification et en évitant les boucles sont développé pour la résolution de l'application.

- Chapitre 5, nous donnons la conclusion et les perspectives.

.1 Introduction

Les réseaux de Petri (RdP) ont été largement utilisés pour la modélisation, l'analyse, la synthèse et l'implémentation de systèmes de production manufacturiers depuis plusieurs décennies [RSET04], [ZV99]. Les systèmes de production manufacturiers sont généralement caractérisés par des occurences d'événements discrets concurrents. Les RdP sont bien adaptés à la modélisation de tels systèmes car ils capturent de manière compacte les relations de précédence et les interactions entre ces événements. Puisque les RdP sont fondés sur une base mathématique solide, ils permettent à la fois des approches d'analyse qualitative et quantitative des propriétés de tels systèmes.

Lorsque l'on utilise les RdP, de nombreux problèmes liés aux systèmes de production manufacturiers peuvent être exprimés comme des problèmes d'*accessibilité*, qui consistent à trouver une séquence de franchissements de transitions menant d'un état initial à un état final. Lorsque les durées des opérations ou des événements doivent être considérés, plusieurs classes de RdP peuvent être utilisés, comme les réseaux de Petri *temporisés* (RdPT), que nous considérons dans cette thèse. Nous considérons les problèmes d'accessibilité dans les RdPT dans deux contextes particuliers : l'ordonnancement de tâches et la reconfiguration de systèmes. Un problème d'ordonnancement peut se traduire par la recherche de la séquence de franchissement optimale entre deux états. Ce problème peut être résolu à l'aide de solveurs de programmation linéaire en nombres entiers. Dans le cas de la reconfiguration, deux problématiques doivent être considérées. La

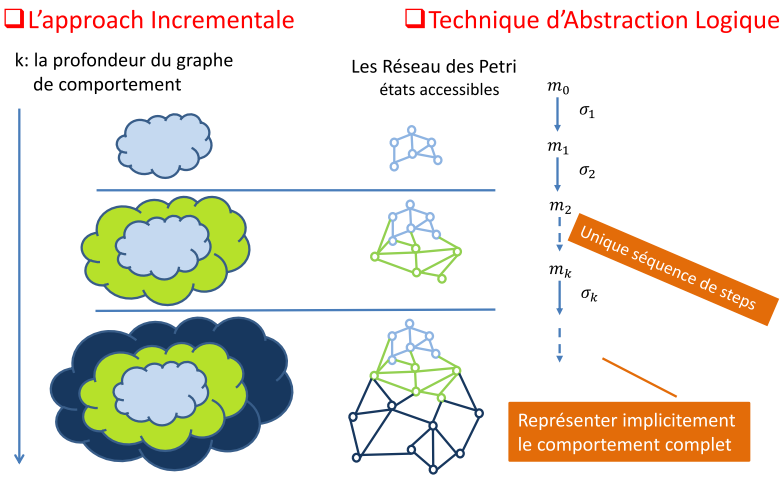


Figure 1: Principales Méthodes

reconfiguration dynamique (en ligne) nécessite de trouver aussi rapidement que possible une séquence d'opérations, pas forcément optimale. L'analyse *hors-ligne* des possibilités de reconfiguration nécessite au contraire d'*énumérer* l'ensemble des séquences possibles. Les solveurs de programmation linéaire étant mal adaptés à ces types de recherche, nous proposons d'évaluer la capacité des solveurs de programmation par contraintes sur ces problèmes.

L'approche proposée se fonde sur un *parcours implicite* du graphe d'accessibilité du RdP, qui ne nécessite pas sa construction effective, comme présenté dans la figure (1). Cela est réalisé par la construction d'une unique *séquence de steps temporisés* de taille croissante pour représenter de façon incrémentale le comportement du RdP considéré. Nous montrons dans cette thèse comment notre approche incrémentale peut être utilisée pour résoudre divers problèmes de production manufacturière modélisés comme des problèmes d'accessibilité dans les RdPT. Le modèle mathématique construit est aussi général que possible puisqu'il ne fait pas d'hypothèses au sujet de la sémantique de franchissement des transitions, contrairement aux autres approches classiques dédiées à la même problématique.

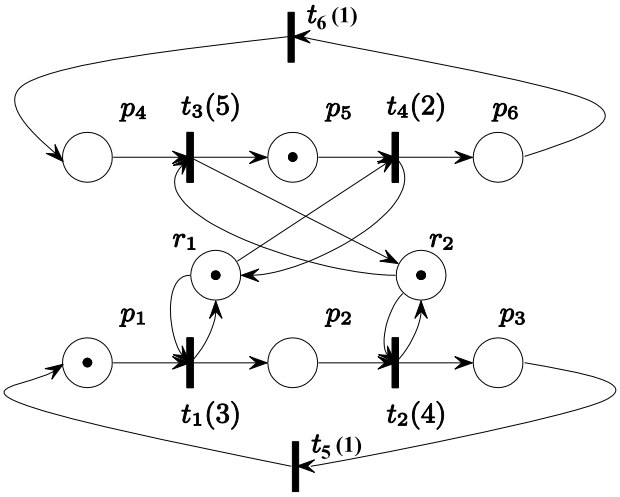


Figure 2: Un Réseau de Petri Temporisé

.2 Approche incrémentale pour les réseaux de Petri temporisés

.2.1 Problème d'accessibilité dans les réseaux de Petri temporisés

Les réseaux de Petri temporisés (RdPT) ont été introduits par [Ram74]. Nous suggérons au lecteur l'article [Mur89] pour une présentation complète des réseaux de Petri. Dans cette thèse, la présentation s'inspire des travaux de [Chr84]. Nous commençons par donner une brêve présentation informelle des RdPT.

.2.1.1 Présentation informelle

Les réseaux de Petri temporisés sont des réseaux de Petri places/transitions (RdP), dans lesquels une *durée* $d(t) \in \mathbb{N}^*$ est associée à chaque transition t. Un RdPT a la même représentation qu'un Réseau de Petri, à laquelle est ajoutée un *étiquetage* des transitions. Un exemple de RdPT est donné dans la figure (2). On a: $d(t_1) = 3$, $d(t_2) = 4$, $d(t_3) = 5$, $d(t_4) = 2$.

Les durées de franchissement associées aux transitions modifient les conditions de validité des marquages. Lors du franchissement d'une transition, tout se passe

	Date	Marquage $(p_1, p_2, \ldots, p_6, r_1, r_2)$
La date initiale	0	$(1,0,0,0,1,0,1,1)$
Tirs $t_1 \rightarrow$	1	$(0,0,0,0,1,0,0,1)$
	2	$(0,0,0,0,1,0,0,1)$
	3	$(0,0,0,0,1,0,0,1)$
Finir $t_1 \rightarrow$	4	$(0,1,0,0,1,0,1,1)$
Tirs $t_4 \rightarrow$	5	$(0,1,0,0,0,0,0,1)$
Tirs $t_2 \rightarrow$	6	$(0,0,0,0,0,0,0,0)$
Finir $t_4 \rightarrow$	7	$(0,0,0,0,0,1,1,0)$
	8	$(0,0,0,0,0,1,1,0)$
	9	$(0,0,0,0,0,1,1,0)$
Finir $t_2 \rightarrow$	10	$(0,0,1,0,0,1,1,1)$
Tirs $t_5 \rightarrow$	11	$(0,0,0,0,0,1,1,1)$
End of $t_5 \rightarrow$	12	$(1,0,0,0,0,1,1,1)$
Tirs $t_1 \rightarrow$	13	$(0,0,0,0,0,1,0,1)$

Table 1: Sémantiques de franchissement de la figure (2)

comme si les jetons *« disparaissaient »* au moment où la transition est franchie, puis *« réapparaissaient »* après un délai correspondant à la durée de la transition en question. Ainsi, le marquage d'un RdPT évolue suivant le rythme d'un horloge externe. Par exemple, considérons la table (1) de la figure (2). À la date 1, la transition t_1 (durée: 3 *u.t.*) est franchie. Puis la transition t_4 (durée: 2 *u.t.*) est franchie à la date 5. L'évolution du marquage en fonction du temps est donnée dans la figure (2). Il faut noter que l'on aurait pu franchir la transition t_4 à la date 4, puisque la ressource r_1 est rendue à la fin du franchissement de t_1. Cependant, la même transition n'était pas franchissable à la date 3, puisque le franchissement de t_1 n'était pas terminé.

Les dates de début et de fin de franchissement jouent un rôle fondamental dans le comportement du RdPT. Il est donc nécessaire d'*adapter* les équations de franchissement en fonction de ces dates particulières. De manière à respecter la sémantique de franchissement sous-jacente des RdP, une séquence de franchissement est dite *franchissable* si et seulement si, à tout instant, le marquage intermédiaire atteint est constitué exclusivement de composantes non négatives.

.2.1.2 Réseaux de Petri temporisés

Definition .2 (RdPT – Réseau de Petri Temporisé). *Un réseau de Petri temporisé [Ram74] $(R = (\mathbb{P}, \mathbb{T}, C^+, C^-), d)$, accompagné de son marquage initial m_0 est un graphe orienté biparti où:*

- *$\mathbb{P}$ et $\mathbb{T}$ sont deux ensembles finis de nœuds dénotés respectivement places et transitions avec $|\mathbb{P}| = \text{M}$ et $|\mathbb{T}| = \text{N}$. Les places sont représentées par des cercles et les transitions par des rectangles. Les places représentent généralement des conditions, et les transitions des événements, d'après [Mur89];*
- *Les matrices d'incidence C^-, C^+ et $C \in \mathbb{N}^{\mathbb{P}\times\mathbb{T}}$ (avec $C = C^+ - C^-$) définissent les relations d'incidence qui associent à chaque arc (p_i, t_j) ou (t_j, p_i) son poids C^-_{ij} or C^+_{ij}. Lorsqu'il n'y a pas d'arc entre une place p_i et une transition t_j, on a : $C^-_{ij} = C^+_{ij} = 0$;*
- *La fonction $m_0 : \mathbb{P} \to \mathbb{N}$ associe à chaque place $p \in \mathbb{P}$ un entier $m_0(p)$ appelé marquage de p. Les marquages sont représentés par des disques pleins appelés jetons à l'intérieur des places;*
- *La fonction $d : \mathbb{T} \to \mathbb{N}^*$ associe à chaque transition du réseau une durée entière strictement positive.*

Dans la suite, nous utilisons des formulations fondées sur l'algèbre linéaire pour plus de concision. Par exemple, $\overrightarrow{e_{t_k}}$ note le vecteur caractéristique associé à une transition t_k, dont la k$^{\text{ième}}$ composante prend la valeur 1 et les autres 0. $\overrightarrow{0_d}$ note un vecteur entièrement nul de dimension d.

Pour simplifier l'étude, nous nous limitons aux RdPT *sans transitions immédiates* (i.e. $\forall t \in \mathbb{T}, d(t) > 0$), ce qui n'est pas très restrictif dans les problèmes réels et correspond bien aux problématiques rencontrées dans les systèmes de production manufacturières.

La sémantique de franchissement des RdPT interdit la *réentrance*, ce qui signifie qu'il n'est pas possible de franchir une nouvelle fois une transition qui est déjà en cours de franchissement (on parle de politique *mono-serveur*). Cette restriction est bien adaptée aux problématiques des systèmes manufacturières où les transitions sont associées à des opérations d'usinage sur des machines. Ainsi, il

est possible d'associer une unique *durée résiduelle* à chaque transition, sans confusion possible entre des activations concurrentes. Le vecteur des durées résiduelles est associé au marquage d'un RdPT pour définir son état complet.

Definition .3 (État d'un RdPT). *Soit (R, d) un RdPT. Son état $s = (\overrightarrow{s_m}, \overrightarrow{s_r})$ est défini par :*

- *le traditionnel vecteur de marquage $\overrightarrow{s_m} \in \mathbb{N}^{\mathrm{M}}$ qui associe à chaque place son nombre de jetons ;*
- *un vecteur de durées résiduelles $\overrightarrow{s_r} \in \mathbb{N}^{\mathrm{N}}$, associant à chaque transition active sa durée résiduelle, ou 0 si la transition n'est pas active.*

L'ensemble des états d'un RdPT est dénoté par $\mathscr{S}(R, d)$. Le concept fondamental qui gouverne le comportement des RdPT est la notion d'*exécution contrôlée*, introduite par [Chr84] qui associe à chaque transition la suite de ses dates de franchissement successives.

Definition .4 (EC – Exécution Contrôlée). *Soit (R, d) un RdPT et $t \in \mathbb{T}$ une transition. Une séquence de franchissement pour la transition t : $(u_k^t) = u_1^t, \ldots, u_{\mathrm{K}_t}^t \in \mathbb{N}$ est une suite croissante de dates de franchissement, telle que :*

$$\forall k \in [\![1, \mathrm{K}_t - 1]\!], u_k^t + d(t) \leqslant u_{k+1}^t \tag{1}$$

Une exécution contrôlée est une famille $(u_k^t)_{t \in \mathbb{T}, k \in [\![1, K_t]\!]}$ de séquences de franchissement pour toutes les transitions du RdPT.

Il faut noter que dans la définition précédente, l'équation (1) est utilisée pour interdire la *réentrance*. Nous dénotons par $v_{\max}$ la date de fin du dernier franchissement de l'EC : $v_{\max} = \max_{t \in \mathbb{T}} \left(u_{\mathrm{K}_t}^t + d(t)\right)$. Après $v_{\max}$, l'état du RdPT soumis à l'exécution contrôlée considérée ne changera plus.

L'expression formelle d'une EC est utilisée pour définir plusieurs *vecteurs caractéristiques* permettant de vérifier sa validité. Nous faisons l'hypothèse qu'aucune transition n'est active à l'instant initial pour simplifier la formulation.

Definition .5 (Vecteur caractéristiques des exécutions contrôlées). *Soit (R, d) un RdPT avec son état initial $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ donné à l'instant 0 et $(u_k^t)_{t \in \mathbb{T}, k \in [\![1, K_t]\!]}$ une EC. Soit $v \in [\![0, v_{\max}]\!]$. Deux vecteurs caractéristiques sont associés à (u_k^t) de la manière suivante :*

- $\overrightarrow{N(v)} \in \mathbb{N}^{\mathrm{N}}$ *est le vecteur correspondant au nombre de franchissements ayant débuté dans l'intervalle* $[0, v]$*, défini par :* $\overrightarrow{N(v)}\Big|_t = card\left(\{u_k^t, k \in [\![1, \mathrm{K}_t]\!] \mid u_k^t \leqslant v\}\right)$*;*
- $\overrightarrow{F(v)} \in \mathbb{N}^{\mathrm{N}}$ *est le vecteur correspondant au nombre de franchissements qui se sont terminés dans l'intervalle* $[0, v]$*, défini par :* $\overrightarrow{F(v)}\Big|_t = card\left(\{u_k^t, k \in [\![1, \mathrm{K}_t]\!] \mid u_k^t + d(t) \leqslant v\}\right)$*.*

L'état d'un RdPT est modifié sous l'action d'une EC de la manière suivante.

Definition .6 (États instantanés d'un RdPT sous l'action d'une EC). *Soit* (R, d) *un RdPT avec son état initial* $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ *donné à la date* 0 *et* $(u_k^t)_{t\in\mathbb{T},k\in[\![1,K_t]\!]}$ *une EC. Soit* $v \in [\![0, v_{\max}]\!]$*. L'état instantané du réseau* $e_v = \left(\overrightarrow{s_m(v)}, \overrightarrow{s_r(v)}\right)$ *à la date* v *est défini par :*

$$\overrightarrow{s_m(v)} = \overrightarrow{s_{m_0}} + C^+ \cdot \overrightarrow{F(v)} - C^- \cdot \overrightarrow{N(v)} \tag{2}$$

$$\forall t \in \mathbb{T}, \overrightarrow{s_r(v)}\Big|_t = \begin{cases} u_k^t + d(t) - v & si\ \exists k \in [\![1, \mathrm{K}_t]\!]\ st\ v \in [\![u_k^t, u_k^t + d(t)[\![\\ 0 & sinon \end{cases} \tag{3}$$

Informellement, dans la définition précédente, la quantité $C^+ \cdot \overrightarrow{F(v)}$ correspond aux jetons produits par les franchissements des transitions qui se sont terminées *jusqu'à la date* v *comprise*. Ces jetons peuvent être utilisées pour franchir des transitions à la date v. La quantité $C^- \cdot \overrightarrow{N(v)}$ correspond aux jetons utilisés par les franchissements des transitions qui ont débuté *jusqu'à la date* v *comprise*. Ainsi, $\overrightarrow{s_m(v)}$ donne exactement le nombre de jetons restants dans le RdPT à la date v.

De manière évidente, comme pour les RdP, même si chaque transition est individuellement franchissable à la date prévue par l'EC, l'exécution contrôlée complète n'est pas nécessairement valide dans son ensemble puisqu'un jeton pourrait être utilisé par plusieurs transitions au même moment. Ainsi, une condition pour qu'une EC soit valide est donnée ci-dessous.

Definition .7 (Exécution contrôlée valide). *Soit* (R, d) *un RdPT avec son état initial* $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ *donné à la date* $0h$ *et* $(u_k^t)_{t\in\mathbb{T},k\in[\![1,K_t]\!]}$ *une EC. Cette*

EC est dite valide ssi :

$$\forall v \in [\![0, v_{\max}]\!], \overrightarrow{s_m(v)} \geqslant \overrightarrow{0_{\mathrm{M}}} \tag{4}$$

La condition précédente signifie qu'il doit y avoir suffisamment de jetons pour que toutes les transitions puissent être franchies simultanément.

.2.1.3 Problème d'accessibilité dans les réseaux de Petri temporisés

En utilisant les notations précédentes, le problème d'accessibilité dans les réseaux de Petri temporisés consiste à rechercher une exécution contrôlée valide permettant d'atteindre un état final donné à partir de l'état initial.

Definition .8 (Problème d'accessibilité dans les RdPT). *Soit (R, d) un RdPT avec son état initial $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ donné à la date 0. Soit $s_f = \left(\overrightarrow{s_{m_f}}, \overrightarrow{0_{\mathrm{N}}}\right)$ un état cible. Le problème d'accessibilité dans les RdPT consiste à trouver une EC $(u_k^t)_{t\in\mathbb{T},k\in[\![1,K_t]\!]}$ telle que $s_{v_{\max}} = \left(\overrightarrow{s_m(v_{\max})}, \overrightarrow{s_r(v_{\max})}\right) = s_f$.*

Plusieurs approches ont été proposées pour résoudre le problème d'accessibilité dans les RdPT, soit en restreignant l'étude à une sous-classe de RdPT, comme les graphes d'événements temporisés, soit par l'utilisation d'heuristiques dédiées. Une bibliographie complète est disponible dans [Ric00].

Puisque le franchissement d'une transition peut intervenir aussitôt qu'elle est sensibilisée mais peut également être différée autant que souhaité, il peut exister, à partir d'un état donné, un nombre infini d'états accessibles (différents du point de vue des dates de franchissement), et aucun graphe d'accessibilité ne peut être construit. Une première approche peut être de considérer les RdPT comme une *sous-classe des RdP temporels*, de manière à utiliser les méthodes énumératives (graphes de classes d'état) proposées par [BD91].

Une autre possibilité est offerte lorsque l'on utilise la sémantique de franchissement au plus tôt (pour laquelle une transition est franchie dès qu'elle devient franchissable). Dans ce cas, il est possible de procéder à des analyses énumératives et structurelles [DA92], et de construire le graphe des marquages au plus tôt [CC88]. Il faut noter que cette approche ne peut être utilisée pour le problème d'accessibilité dans le cas général car franchir les transitions aussitôt que possible peut conduire à manquer des séquences de franchissement optimales.

La sémantique au plus tôt a été largement étudiée pour la classe particulière des graphes d'événements temporisés (où une place a exactement une transition d'entrée et une transition de sortie), en utilisant l'algèbre $(max, +)$ [BCOQ92]. Puisque leur structure ne contient pas de transitions en conflit, il est possible d'obtenir des équations linéaires permettant de représenter le comportement complet du réseau.

Une solution plus générale peut être trouvée dans [DYKG04], où des séquences de franchissement non temporisées sont d'abord générées, puis associées à des dates de tir en considérant alors uniquement les contraintes temporelles. Malheureusement, une telle méthode nécessite d'énumérer d'abord toutes les séquences, ce qui mène à des problèmes d'explosion combinatoire, et il peut ensuite exister une infinité d'associations d'instants de tir aux transitions (si une transition peut être tirée à l'instant t, elle peut aussi l'être à tout moment plus tard).

Dans la suite, nous montrons que notre approche incrémentale permet de générer des modèles de programmation par contraintes dans le cadre le plus général.

.2.2 Approche incrémentale

.2.2.1 Présentation informelle

Dans cette section, nous proposons de bâtir un modèle mathématique de façon *incrémentale*, fondé sur l'expression des contraintes correspondant à un *nombre croissant d'instants de tir*.

Contrairement aux exécutions contrôlées, il n'est pas nécessaire de considérer les marquages intermédiaires entre deux états pour vérifier la validité d'une séquence de steps. En effet : même si une transition qui a débuté à la date v_i a terminé son franchissement à une date strictement inférieure à la date de franchissement suivante v_{i+1}, ses jetons ont déjà été consommés à la date v_{i+1}, de telle sorte qu'il suffit que les marquages soient positifs à chaque instant de tir pour que les marquages intermédiaires entre deux instants de tir le soient aussi. De telles considérations permettent de réduire le nombre d'états à considérer, et permettent donc de réduire l'explosion combinatoire.

La principale idée dans notre formulation consiste à étudier l'évolution du RdPT *"step par step"*. L'expression *step* est empruntée à [JK91].

s_{m_0}	$\longrightarrow$	$s_{m_1} \dots s_{m_k}$	$\longrightarrow$	$s_{m_{k+1}} \dots s_{m_{\mathsf{K}-2}}$	$\longrightarrow$	$s_{m_{\mathsf{K}-1}}$	$\longrightarrow$	$s_{m_{\mathsf{K}}} = s_{m_f}$
	σ_1		σ_{k+1}		$\sigma_{\mathsf{K}-1}$		$\sigma_{\mathsf{K}} = 0_N$	
$v_0 = 0$	$v_1 =$		$v_{k+1} =$		$v_{\mathsf{K}-1} =$		$v_{max} =$	
	$v_0 + \Delta_{v_0}$		$v_k + \Delta_{v_k}$		$v_{\mathsf{K}-2} + \Delta_{\mathsf{K}-2}$		$v_{\mathsf{K}-1} + \Delta_{\mathsf{K}-1}$	
s_{r_0}	$\longrightarrow$	$s_{r_1} \dots s_{r_k}$	$\longrightarrow$	$s_{r_{k+1}} \dots s_{r_{\mathsf{K}-2}}$	$\longrightarrow$	$s_{r_{\mathsf{K}-1}}$	$\longrightarrow$	$s_{r_{\mathsf{K}}} = s_{r_f}$

Table 2: Tableau Synoptique des Notations.

Un schéma synoptique des notations est donné dans la table (2). Nous exprimons à chaque instant de franchissement v_{k+1} (défini par le délai Δ_{v_k} écoulé depuis le franchissement précédent) les modifications induites sur les vecteurs $\overrightarrow{s_m}(v_{k+1})$ et $\overrightarrow{s_r}(v_{k+1})$ par le franchissement du step $\sigma(v_{k+1})$. De manière à être plus concis, nous dénotons les expresions intermédiaires en utilisant l'indice de la date de franchissement correspondante : $\overrightarrow{s_m}(v_k) \equiv \overrightarrow{s_{m_k}}$, $\sigma(v_k) \equiv \sigma_k$, etc.

.2.2 Steps temporisés

Definition .9 (ST – Step Temporisé). *Soit (R, d) un RdPT. Un step temporisé est la donnée d'une paire (σ_k, v_k) telle que :*

- $v_k \in \mathbb{N}$;
- $\sigma_k \in \{0,1\}^{\mathbb{T}}$ *est une fonction dénotant un ensemble de franchissements concurrents à la date v, sous l'hypothèse d'une politique mono-serveur. On note $\overrightarrow{\sigma_k}$ le vecteur caractéristique des nouvelles transitions franchies à la date v_k.*

Pour exprimer le comportement d'un RdPT entre deux dates v_k et v_{k+1} sous la forme d'une équation d'état, nous définissons deux ensembles de transitions dont les franchissements *se terminent* ou *se poursuivent* à la date v_k.

Definition .10 (Ensembles de franchissements). *Soit (R, d) un RdPT avec son état initial $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathsf{N}}}\right)$ donné à la date v_0. Soit $(\sigma_i, v_i)_{i \in [\![1,k]\!]}$ une séquence de k steps temporisés et $v_{i+1} \in \mathbb{N}$ t.q. $v_{i+1} > v_i$. Les ensembles de franchissements T^f_{k+1} et T^s_{k+1} sont définis à la date v_{k+1} de la manière suivante :*

- *L'ensemble des transitions se terminant $T^f_{k+1} \in \mathbb{T}^{\mathsf{N}}$ (« f » pour « finishing ») note les transitions actives à la date v_k qui ne sont plus en cours de*

franchissement à la date v_{k+1}. Formellement :

$$T^f_{k+1}=\Big\{ \quad t \in \mathbb{T}, \exists i \in [\![1,k]\!] \text{ s.t. } \sigma_i(t)=1 \\ \wedge \quad v_k < \max_{i\in[\![1,k]\!]}\{v_i : \sigma_i(t)=1\} + d(t) \leqslant v_{k+1} \Big\} \tag{5}$$

- *L'ensemble des transitions se poursuivant $T^s_{k+1} \in \mathbb{T}^{\mathsf{N}}$ (« s » pour « still firing ») note les transitions actives à la date v_k qui sont toujours en cours de franchissement à la date v_{k+1}. Formellement :*

$$T^s_{k+1}=\Big\{ \quad t \in \mathbb{T}, \exists i \in [\![1,k]\!] \text{ s.t. } \sigma_i(t)=1 \\ \wedge \quad v_k < v_{k+1} < \max_{i\in[\![1,k]\!]}\{v_i : \sigma_i(t)=1\} + d(t) \Big\} \tag{6}$$

A l'aide de ces notations, l'équation d'état des RdPT utilisant des steps temporisés peut s'exprimer de la façon suivante : pour qu'un step temporisé soit franchissable, son marquage de départ doit contenir assez de jetons pour que chaque transition du step consomme ses *propres* jetons, comme l'indique la définition ci-dessous.

Definition .11 (Franchissement de steps temporisés). *Soit (R,d) un RdPT avec son état initial $s_0 = \left(\overrightarrow{s_{m_0}}, \overrightarrow{0_{\mathrm{N}}}\right)$ donné à la date v_0. Soit $(\sigma_i, v_i)_{i\in[\![1,k]\!]}$ une séquence de k steps temporisés menant à l'état $s_k = (\overrightarrow{s_{m_k}}, \overrightarrow{s_{r_k}})$ à la date v_k. Soit $v_{k+1} \in \mathbb{N}$ t.q. $\Delta_k = v_{k+1} - v_k > 0$. Le step temporisé (σ_{k+1}, v_{k+1}) est franchissable depuis s_k ssi:*

$$\forall t \in \mathbb{T} \text{ t.q. } \sigma_{k+1}(t)=1, \quad \overrightarrow{s_{r_k}}(t) \leqslant \Delta_k \tag{7}$$

$$C^- \cdot \overrightarrow{\sigma_{k+1}} \leqslant \overrightarrow{s_{m_k}} + C^+ \cdot \sum_{t_j \in T^f_{k+1}} \overrightarrow{e_{t_j}} \tag{8}$$

Si cette condition est satisfaite, le nouvel état $s_{k+1} = (\overrightarrow{s_{m_{k+1}}}, \overrightarrow{s_{r_{k+1}}})$ atteint à la date v_{k+1} depuis s_k par le franchissement de (σ_{k+1}, v_{k+1}) est défini par :

$$\overrightarrow{s_{m_{k+1}}} = \overrightarrow{s_{m_k}} - C^- \cdot \overrightarrow{\sigma_{k+1}} + C^+ \cdot \sum_{t_j \in T^f_{k+1}} \overrightarrow{e_{t_j}} \tag{9}$$

$$\overrightarrow{s_{r_{k+1}}} = \sum_{t_j \in T^s_{k+1}} \left(s_{r_k}(t) - \Delta_k\right) \cdot \overrightarrow{e_{t_j}} + \sum_{t_j \in \mathbb{T}} d(t_j) \cdot \sigma_{k+1}(j) \cdot \overrightarrow{e_{t_j}} \tag{10}$$

La définition ci-dessus suit la sémantique de franchissement des RdPT décrite plus tôt, du point de vue d'un franchissement ponctuel entre deux états. L'expression $C^- \cdot \overrightarrow{\sigma_{k+1}}$ représente le nombre de jetons extraits des places en amont des transitions nouvellement franchies σ_{k+1} à la date v_{k+1}. L'expression $C^+ \cdot \sum_{t_j \in T^f_{k+1}} \overrightarrow{e_{t_j}}$ représente les jetons ajoutés aux places en aval des transitions qui sont terminées à la date v_{k+1}. L'expression $\sum_{t_j \in T^s_{k+1}} \left(s_{r_k}(t) \ - \Delta_k\right) \cdot \overrightarrow{e_{t_j}}$ représente la mise à jour des durées résiduelles des transitions qui étaient actives à la date v_k et sont toujours actives à la date v_{k+1}. La durée écoulée correspond à $\Delta_k = v_{k+1} - v_k$ unités de temps. L'expression $\sum_{t_j \in \mathbb{T}} d(t_j) \cdot \sigma_{k+1}(j) \cdot \overrightarrow{e_{t_j}}$ représente les *nouvelles durées résiduelles* résultant des nouveaux franchissements dans σ_{k+1} à la date v_{k+1}. Finalement, l'équation (7) exprime qu'une transition franchie dans un step ne devait pas être active au moment du franchissement considéré, de manière à respecter la politique mono-serveur.

.2.2.3 Vers un modèle incrémental

Nous donnons ci-dessous les principales propositions concernant l'utilisation des steps temporisés dans le contexte des RdPT.

Proposition .12 (Equivalence entre ECs et séquences de steps temporisés). *Soit (R, d) un RdPT avec son état initial s_0 donné à la date v_0 et un état $s_f \in \mathscr{S}(R, d)$. L'équivalence suivante est vérifiée :*

Il existe une exécution contrôlée valide permettant d'atteindre s_f à la date $v_{\max}$ à partir de s_0.	$\Leftrightarrow$	*Il existe une séquence de steps temporisés franchissable permettant d'atteindre s_f à la date $v_{\max}$ à partir de s_0.*

Corollaire .13 (Capture du comportement d'un RdPT par une séquence de steps temporisés)

Soit (R, d) un RdPT avec son état initial s_0 donné à la date $v_0 = 0$.

Toute exécution contrôlée peut être représentée comme une séquence de steps temporisés. Ainsi, tout état s_f accessible depuis s_0 peut être atteint par une séquence de steps temporisés. La longueur d'une telle séquence dépend de l'état final à atteindre s_f.

La preuve complète de ces propositions est trop longue pour figurer dans cette thèse, et est donnée dans l'appendix A. Les résultats sont assez évidents dans la mesure où les deux formulations suivent la sémantique sous-jacente des RdP ordinaires, et où les ensembles T_k^f et T_k^s utilisés dans la définition (.10) sont directement reliés aux vecteurs $\overrightarrow{N(v)}$ et $\overrightarrow{F(v)}$ utilisés dans la définition (.5). En effet :

$T_{k+1}^f = \left\{t \in \mathbb{T} : \overrightarrow{F(v_k)}\Big|_t = \overrightarrow{N(v_k)}\Big|_t - 1 \ \wedge \ \overrightarrow{F(v_{k+1})}\Big|_t = \overrightarrow{N(v_k)}\Big|_t\right\}$ et $T_{k+1}^s = \left\{t \in \mathbb{T} : \overrightarrow{F(v_k)}\Big|_t = \overrightarrow{N(v_k)}\Big|_t - 1 \ \wedge \ \overrightarrow{F(v_{k+1})}\Big|_t = \overrightarrow{F(v_{k+1})}\Big|_t\right\}$.

On peut remarquer dans les précédentes équations que les ensembles de franchissement à la date $k+1$ peuvent être complètement exprimés à partir de la situation donnée à la date k. Nous montrons ci-dessous comment exprimer T_{k+1}^s et T_{k+1}^f en utilisant uniquement T_k^s, T_k^f, Δ_k et σ_k, de manière à simplifier l'expression du comportement du RdPT, et à la rendre complètement *incrémentale* : il devient alors possible d'augmenter progressivement le nombre de steps temporisés utilisés dans la formulation sans redéfinir l'ensemble complet des contraintes précédemment exprimées.

D'après les précédentes propositions, il est suffisant de rechercher une séquence de steps temporisés pour résoudre le problème d'accessibilité dans les RdPT. L'avantage d'utiliser des steps temporisés est que cela permet de réduire le nombre de franchissements dans le modèle – et donc le nombre de variables – tout en maintenant une équivalence avec le comportement initial. Ainsi, il ne s'agit pas d'une modification de la sémantique des RdPT, mais seulement d'une manière de capturer l'indépendance des transitions. Bien entendu, cette réduction n'intervient pas systématiquement, car il est facile de construire un RdPT dans lequel une seule transition pourrait être franchie à la fois. Cela signifierait cependant que le réseau de Petri ne montre aucun parallélisme, ce qui est assez inhabituel.

Dans notre travail, nous nous intéressons plus particulièrement aux problématiques de sûreté de fonctionnement. La vérification d'une exigence de sécurité peut être effectuée en procédant à la recherche d'une séquence de franchissements temporisés représentant un contre-exemple de la propriété à garantir, ou à l'énumération de l'ensemble des chemins entre deux états pour vérifier que la propriété est toujours vraie. Nous proposons donc d'utiliser une approche fondée sur la programmation par contraintes qui est bien adaptée à ce type de démarche.

.3 Analyse des réseaux de Petri temporisé en utilisant la programmation par contraintes

Dans ce chapitre, nous avons traduit notre modèle incrémental des RdPT en utilisant la programmation par contraintes (PPC). Dans ce cour, la technique la plus importante est d'utiliser les contraintes conditionnelles pour représenter les équations non linéaires des Ensembles de franchissements. La PPC utilise principalement deux étapes pour résoudre le problème: ***Modélisation*** et ***Résolution***. Dans la phrase de *Modélisation*, la PPC peut réduire *l'espace de recherche* formé par les variables en ajoutant quelques améliorations, par exemple, en réduisant les domaines des variables et en ajoutant des nouvelles contraintes. Dans la phrase de *Résolution*, certaines stratégies de recherche développés peuvent trouver une solution en parcourant *l'arbre de recherche* plus efficace, car les nœuds ou des éléments non relevent peuvent être éliminés lors de la recherche. Ce processus de recherche peut réduire l'influence de l'explosion combinatoire d'une manière contrôlée. Dans le meilleur des cas, la PPC peut trouver une solution sans nœuds échouer ce qui signifie sans développer d'autres etats non relevent dans le graphe d'accessibilité.

.3.1 Modèle mathématique pour la programmation par contraintes

Dans cette section, nous commençons par exprimer les franchissements de séquences de steps sous forme d'équations linéaires et de contraintes conditionnelles, classiquement utilisées dans le cadre des approches de type programmation par contraintes.

.3.1.1 Mise à jour des marquages

Pour exprimer la quantité de jetons ajoutés à $s_{m(k+1)}$ par les franchissements de transitions qui se sont terminés entre les dates v_k et v_{k+1}, nous ajoutons les variables intermédiaires $TF_{(k+1)j}$ dénotant les transitions en question, contraintes par les équations suivantes :

$$\forall j \in [\![1, \mathrm{N}]\!], \forall k \in [\![0, \mathrm{K}-1]\!]$$

$$\begin{cases} s_{r_{kj}} > 0 \\ \wedge \quad s_{r_{kj}} - \Delta_{v_k} \leqslant 0 \end{cases} \Rightarrow T^f_{(k+1)j} = 1 \tag{11}$$

$$\begin{cases} s_{r_{kj}} \leqslant 0 \\ \vee \quad s_{r_{kj}} - \Delta_k > 0 \end{cases} \Rightarrow T^f_{(k+1)j} = 0 \tag{12}$$

L'équation (11) permet d'identifier les transitions j qui étaient en cours de franchissement à la date v_k et dont les franchissements sont terminés à la date v_{k+1}. Inversement, l'équation (12) identifie les transitions qui n'étaient pas en cours de franchissement à la date v_k ou qui sont déjà terminées à la date v_{k+1}.

Le vecteur $\overrightarrow{TF_{k+1}}$ est alors utilisé pour exprimer l'équation (9) comme suit :

$$\overrightarrow{s_{m_{k+1}}} = \overrightarrow{s_{m_k}} - C^- \cdot \overrightarrow{\sigma_{k+1}} + C^+ \cdot \overrightarrow{T^f_{k+1}} \tag{13}$$

.3.1.2 Mise à jour des vecteurs de durées résiduelles

Pour exprimer l'équation (10), nous ne faisons pas apparaître explicitement un vecteur caractérisant l'ensemble T^s_{k+1}, mais nous utilisons directement les valeurs de T^s_k pour vérifier si des transitions précédemment actives sont toujours en cours.

Puisque la réentrance est interdite, une transition t_j qui était déjà active à la date v_k ne peut démarrer un nouveau franchissement au step v_{k+1} si le précédent franchissement n'est pas terminé auparavant. Ainsi, à la date v_{k+1}, la composante du vecteur de durées résiduelles correspondant à la transition considérée, $s_{r(k+1)j}$, peut prendre deux valeurs :

$$s_{r_{kj}} - \Delta_k > 0 \Rightarrow \begin{cases} s_{r_{(k+1)j}} = s_{r_{kj}} - \Delta_k \\ \sigma_{(k+1)j} = 0 \end{cases} \tag{14}$$

$$s_{r_{kj}} - \Delta_k \leqslant 0 \Rightarrow s_{r_{(k+1)j}} = d(j) \cdot \sigma_{(k+1)j} \tag{15}$$

L'équation (14) impose que si t_j est toujours active, elle ne peut être franchie une nouvelle fois et que $s_{r(k+1)j}$ doit être mis à jour avec la durée de franchissement restante, compte tenu du temps écoulé. D'après l'équation (15), si t_j n'est pas en cours de franchissement, ou est terminée, à la datev_{k+1}, la valeur de $s_{r(k+1)j}$ est

fonction du nouveau step $\sigma_{(k+1)j}$. Les équations (14) et (15) sont utilisées pour exprimer l'équation (10) dans le modèle final.

.3.1.3 Expression des conditions de franchissement

La politique mono-serveur imposée par l'équation (7) est déjà garantie par la seconde partie de l'équation (14).

De la même manière, l'expression de la condition de franchissement (8) est une conséquence de l'équation (13), puisque les variables du vecteur $\overrightarrow{s_{m_{k+1}}}$ sont contraintes à être positives.

.3.2 Améliorations du modèle

De manière à réduire l'espace de recherche et améliorer l'efficacité de notre modèle de programmation par contraintes, nous réduisons le domaine des variables et ajoutons des contraintes, en fonction des spécificités des RdPT considérés.

.3.2.1 Conditions de franchissement

Puisque toutes les transitions ont une durée de franchissement finie, il n'est pas nécessaire d'attendre après le franchissement du step σ_k à la date v_k plus du temps nécessaire pour que toutes les transitions actives se terminent, c'est-à-dire au plus la composante maximale du vecteur de durées résiduelles s_{rk}, exprimée par $\max_{j\in[\![1,\mathrm{N}]\!]}(\overrightarrow{s_{rkj}})$. En effet, après la date $v_k + \max_{j\in[\![1,\mathrm{N}]\!]}(\overrightarrow{s_{rk}})$, tous les jetons nécessaires pour franchir les transitions actives à la date v_k sont de nouveau disponibles.

En utilisant le même raisonnement, on remarque que les variables Δ_k et s_{rkj} ne dépasseront jamais la plus grande durée de franchissement possible dénotée par $D_{max} = \max_{t\in\mathbb{T}} d(t)$. Ces considérations mènent à l'ajout des contraintes suivantes :

$$\forall k \in [\![0,\mathrm{K}]\!], \forall j \in [\![1,\mathrm{N}]\!],\ s_{r_{kj}} \in \{0, D_{max}\} \tag{16}$$
$$\forall k \in [\![0,\mathrm{K}]\!], \forall j \in [\![1,\mathrm{N}]\!],\ s_{r_{kj}} \leqslant\ d_j \tag{17}$$
$$\forall k \in [\![0,\mathrm{K}-1]\!],\ \Delta_{v_k} \in \{0, D_{max}\} \tag{18}$$
$$\forall k \in [\![0,\mathrm{K}-1]\!],\ \Delta_{v_k} \leqslant max(\overrightarrow{s_{r_k}}) \tag{19}$$

De manière à réduire l'espace de recherche, nous interdisons le franchissement de steps vides, c'est-à-dire de steps ne contenant aucun franchissement de tran-

sition, qui étaient autorisés dans des travaux précédents. Cela permet d'éviter de trouver des solutions bâties à partir d'une autre par le simple ajout d'un step vide. Seul le dernier step est autorisé à être vide, de manière à pouvoir atteindre un état final pour lequel le marquage cible est atteint et plus aucune transition n'est encore active.

Il faut noter que cela implique de choisir le nombre de steps de manière précise puisque si l'on choisit plus de steps que nécessaire, il se peut qu'aucune solution ne soit trouvée. Au contraire, autoriser les steps vides permet d'utiliser une recherche dichotomique pour trouver le nombre minimal de steps nécessaires pour atteindre l'état final.

De manière à éviter les solutions bâties l'une depuis l'autre en découpant un step en plusieurs franchissements individuels successifs à la même date, nous interdisons au délai Δ_k entre deux steps σ_k et σ_{k+1} d'être nul, excepté pour le premier franchissement Δ_0. Ainsi, si une solution existe pour laquelle deux transitions t_i et t_j doivent être franchies simultanément à la date v_k, elles seront franchies dans un unique step $\overrightarrow{\sigma_k} = \overrightarrow{t_j} + \overrightarrow{t_k}$ et pas en utilisant deux steps $\overrightarrow{\sigma_k} = \overrightarrow{t_j}$ et $\overrightarrow{\sigma_{k+1}} = \overrightarrow{t_k}$ avec $\Delta_k = 0$.

Les précédentes améliorations permettent de réduire l'espace de recherche en éliminant des solutions qui seront trouvées de toute façon sous une autre forme. Les contraintes correspondantes sont données dans les équations (20) et (21).

$$\forall k \in [\![0, \mathrm{K}-1]\!], \sum_{j=0}^{\mathrm{N}-1} \sigma_{kj} \neq 0 \tag{20}$$

$$\forall k \in [\![1, \mathrm{K}-1]\!], \Delta_{v_k} \neq 0 \tag{21}$$

.3.2.2 Propriétés structurelles

Puisque les RdP expriment le comportement de systèmes physiques, ils présentent généralement un comportement infini, fait d'exécutions cycliques, mais leur nombre total d'états est borné. Ainsi, il est souvent possible de calculer une borne maximale pour le marquage de chaque place, ce qui permet de réduire le domaine des variables s_{mik} dans le modèle considéré, comme indiqué dans l'équation (22). Ces bornes peuvent être calculées par la génération des P-flots du réseau, comme indiqué dans [KJ87].

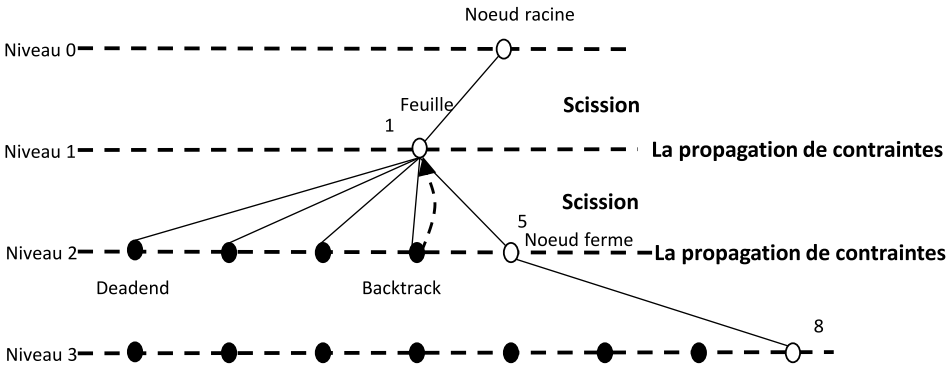


Figure 3: L'arbre de Recherche

$$\forall k \in [\![0, \text{K}]\!], \forall i \in [\![1, \text{M}]\!],\ s_{m_{ki}} \in [\![0, M_{max}]\!] \tag{22}$$

.3.3 Stratégies de recherche

S'il existe une solution, nous pouvons trouver un K pour la trouver. Mais le temps nécessaire pour trouver une solution varie considérablement en fonction des stratégies de recherche et de l'espace de recherche. En général, un arbre de recherche est exploré pendant le processus de recherche, comme presenté par la figure (3). Pour explorer efficacement l'arbre de recherche, plusieurs techniques sont appliquées dans cette thèse, comme l'ordre de prise en compte des variables et de leurs valeurs, et le backtracking.

.3.3.1 Variables

Pour énumérer les variables, les stratégies *génériques* sont donnés par des techniques de PPC et des stratégies *dédiées* sont développés sur la connaissance du comportement des RdPT et de la signification des variables. Dans Ilog Solver, Il y a quelque des stratégies *génériques*, qui sont basées sur la taille du domaine des variables. On peut citer dans le manuel [IBM10].

Mais les stratégies *génériques* ne considèrent pas les informations de la structure des RdPT. Aussi, elles ne donnent pas forcément les meilleures stratégies de recherche, elles ne nous permettent pas de contrôler l'exploration de l'espace de

recherche. Donc, dans cette thèse, nous proposons également de développer des stratégies *dédiées* basées sur notre connaissance du comportement des RdPT et la signification des variables. Elles sont principalement basées sur

- Les steps de franchissement ($\overrightarrow{\sigma_k}$)
- Les dates de franchissement (Δ_{v_k})
- Les marquages ($\overrightarrow{s_{m_k}}$)
- Les vecteurs de durée résiduelle ($\overrightarrow{s_{r_k}}$)

.3.3.2 Valeur

Après avoir choisi une variable à un niveau de l'arbre de recherche, les valeurs de cette variable sont énumérées une par une.

En général, les valeurs d'un intervalle sont énumérées de la borne l'inferieure à la borne superieure. Mais dans notre approche, nous les énumérons dans l'ordre inverse car cette stratégie sera plus efficace et prendre plus de sens physique. Par exemple, $\sigma_{kj} \in [0, 1]$, si $\sigma_{kj} = 1$, cela signifie que la transition t_j sera tirée à la date v_k, autrement $\sigma_{kj} = 0$. On va perdre beaucoup de temps sur le franchissement d'un σ_{kj} vide. Par exemple, la solution ϕ_1, o_{31} est tiré première, et séquence de steps temporisés sera comprimé dans la dernière step.

.3.3.3 Backtrack

En général, les principaux algorithmes de backtrack sont dérivés de [IBM10]. Il y a les algorithmes comme, Depth-First Search (DFS), Best First Search (BFS), Slice-Based Search (SBS) etc.

Dans cette thèse, nous avons testé beaucoup des benchmarks et essayé beaucoup des stratégies de recherche. Basé sur les benchmarks, nous pouvons conclure que la strategie la plus important est la *step-by-step* qui est la plus proche de sens physique du RdPT. Pendant ce temps, énumére variables relatives (σ_k et s_{mk} de haut en bas, et Δ_k et s_{rk} de bas en haut) améliorera beaucoup l'efficacité.

.4 Application

Dans cet chapitre, nous appliquons notre approche incrémentale à la reconfiguration des systèmes de production manufacturière et nous montrons son efficacité.

.4.1 Reconfiguration

Dans cette section, les principaux problèmes des systèmes de fabrication sont d'abord introduits en utilisant un *système de transport reconfigurable*. Le processus de reconfiguration consiste en deux étapes: *la prise de décision* et *la mise en œuvre opérationnelle des actions reconfiguration*.

La prise de décision consiste à déterminer un nouvel état objectif à atteindre. Le processus opérationnel consiste à déterminer la *procédure à appliquer* pour atteindre cet état objectif à partir de l'état courant. Ensuite, la méthodologie pour les systèmes de production est basée sur les RdPT. Le modèle RdPT est principalement constitué du *Pregraph* et *des Gammes Opératoires Étendues*. Ils décrivent respectivement les solutions de routage et l'ensemble des combinaisons d'opérations d'usinage pour produire des pièces finies. En outre, des modèles supplémentaires sont introduit pour effectuer la reconfiguration de systèmes de production.

.4.2 Identification des jetons

Lorsque plusieurs produits sont simultanément transportés, il peut y avoir une confusion de jetons dans le *Pregraph*. Par conséquent des techniques d'identification du jeton pour RdP ordinaires sont présentées dans [HBT07]. Á chaque place, on associe un nombre entier non-négatif appelé *ID* du jeton. Cette technique d'identification du jeton peut être bien sûr utilisée pour tout RdP sauf.

Comme les jetons sont absents pendant le franchissement de la transition, un nouveau vecteur $\overrightarrow{s_{R_d}}$ est défini pour enregistrer les identificateurs des jetons manquants correspondants. Cette méthode est limitée aux RdPT saufs, puisque deux identificateurs de jetons ne peuvent pas être confondus dans le même place.

Dans ce travail, nous avons également proposé une *technique d'identification du jeton pour les RdPT bornés*. Pour cela, nous proposons d'associer un vecteur

d'identificateurs pour chaque type de jeton (jetons avec le même *ID*) que nous nommons respectivement $\overrightarrow{s_{Id_x}}$ et $\overrightarrow{s_{Rd_x}}$.

Cette méthode peut réduire la complexité de la séparation et de la mise à jour des identificateurs de jetons entre les places et transitions, puisque chaque vector d'identificateur de jeton semble être mis à jour indépendamment dans son réseau seul. Ainsi, nous définissons ce réseau partiel d'indépendant *la couche d'identification des jetons dans les RdPT*. Et il peut facilement exprimer le comportement du RdPT avec plusieurs jetons d'identifiants différents.

Bien que l'espace de recherche semble être élargi par la couche de jeton d'identification, les identificateurs de jetons doivent suivre le marquage des RdPT originaux. Par conséquent, l'efficacité de la recherche d'une solution est presque la même que celle du RdPT original. Ensuite, nous comparons ces techniques d'identification du jeton en utilisant le même exemple pour comparer l'efficacité relative de nos méthodes.

Mais lorsque l'on étudie le système de transport reconfigurable, nous avons découvert que les solutions restent difficiles à générer ne sont guère trouvé en raison de boucles dans le *Pregraph*, car ils conduisent le Solver à aller profondément dans l'arbre de recherche.

.4.3 Éviter les boucles

Nous proposons deux types de méthodes pour éviter l'influence de boucles basées sur les réseaux d'identificateurs de jetons dans le RdPT. En premier lieu, les contraintes conditionnelles sur la base de la structure du *Pregraph* sont introduites pour éviter les boucles des robots. Même si cette méthode peut effectivement conduire le Solver à obtenir des solutions, cette méthode est très spécifique à ce modèle.

Deuxièmement, nous développons un nouveau vecteur – priorité de franchissement $\overrightarrow{FP}$ – pour les transitions dans le *Pregraph*. L'élément de $\overrightarrow{FP}$ montre la priorité d'une transition et sera mis en 0 après le franchissement.

Par conséquent, les jetons ne peuvent pas tirer la même transition à nouveau, ce qui peut aider à éviter les boucles. Comme $\overrightarrow{FP}$ est définie sur chaque identificateur de jeton, les jetons avec des ID différents ne seront pas interdits pour la même transition. Si les jetons ont besoin d'aller dans certaines boucles, $\overrightarrow{FP}$ peut

être remis à zéro en fonction de ces besoins.

Enfin, des benchmarks présentent l'efficacité de nos approches.

.5 Conclusion et perspectives

Dans notre thèse, nous nous intéressons à la génération de séquences de franchissement dans les RdPT. Notre objectif est de développer des méthodes efficaces permettant d'adresser les problématiques de sûreté dans les systèmes de production et de transport, par l'énumération de séquence de franchissement *représentatives*.

Nous proposons d'utiliser un modèle construit incrémentalement pour capturer le comportement d'un RdPT sous la forme d'un système d'équations linéaires et de contraintes conditionnelles. Un modèle de programmation par contraintes est ainsi proposé, ainsi que des enrichissements visant à améliorer son efficacité.

Ce modèle a été évalué sur des exemples académiques avec des résultas prometteurs dans [HBYT12b]. Des techniques de linéarisation permettant d'éviter d'utiliser des contraintes conditionnelles ont été développées dans [HBYT12c], avec une étude sur différentes stratégies d'énumération.

Plus récemment, nous avons proposé d'améliorer l'expressivité de notre modèle dans [HBYT12a], en ajoutant la prise en charge d'une couche d'*identification des jetons* au RdPT.

Bien entendu, nous ne prétendons pas que l'explosion combinatoire a été repoussée. Elle est seulement déplacée dans la phase de résolution des contraintes. Cependant, notre formulation permet de bénéficier automatiquement des progrès des meilleurs solveurs actuels. Un autre intérêt de cette formulation est d'éviter l'exploration des branches du graphe d'accessibilité qui ne conduisent pas au marquage recherché. Ces deux derniers points nous font penser que notre méthode peut apporter des améliorations au domaine de recherche considéré.

Dans le futur, nous proposons de poursuivre les intéressantes perspectives de recherche soulevées par ce travail :

- compléter les expérimentations numériques en appliquant notre méthodologie à des applications de taille réelle ;

- améliorer le modèle obtenu en ajoutant d'autres bornes, inégalités valides, contraintes globales traduisant des propriétés structurelles du RdPT ;
- adapter les techniques d'exploration de la programmation par contraintes aux spécificités du problème considéré, en définissant des techniques de filtrage et en améliorant les techniques d'énumération.

Finalement, nous devons signaler que l'appproche incrémentale décrite ici ne mène qu'à des algorithmes de semi-décision, dans la mesure où le nombre d'instants de tir K est défini de manière arbitraire, puisque nous n'avons pas d'information sur le nombre de steps nécessaires pour éventuellement trouver une solution. Ainsi, si aucune solution n'est trouvée pour cette valeur de K, on ne peut conclure sur la possibilité d'atteindre l'état final ou non.

Pourtant, lorsque l'on étudie des RdPT bornés, il est possible d'affecter à K la valeur de la *profondeur séquentielle* du réseau, un paramètre défini dans [BHY04] qui garantit l'exploration complète du graphe d'accessibilité. En utilisant ce paramètre comme profondeur de recherche, il est toujours possible de conclure lorsque l'algorithme s'arrête. Une direction de recherche prometteuse serait de définir des procédures de recherche efficaces pour calculer la valeur de ce paramètre.

Bibliography

[ADL12] S. Amari, I. Demongodin, and J. J. Loiseau. Max-plus control design for temporal constraints meeting in timed event graphs. *Automatic Control, IEEE Transactions on*, 57(2):267–272, 2012. 18

[AK77] T. Araki and T. Kasami. Some decision problems related to the reachability problem for petri nets. *Theoretical Computer Science*, 3:85–104, 1977. 9

[Ake78] S.B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27(6):509–516, juin 1978. 9

[AL04] M.R. Abdi and A.W. Labib. Grouping and selecting products: the design key of reconfigurable manufacturing systems. *International Journal of Production Resources*, 42(32):521–546, 2004. 19

[Apt03] K. Apt. *Principles of constraint programming*. Cambridge University Press, 2003. 59, 62, 64

[BCC+03] A. Biere, A. Cimatti, E.M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in computers*, 58(99):117–148, 2003. 21

[BCCZ99] Armin. Biere, Alessandro. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207, London, UK, UK, 1999. Springer-Verlag. 20, 22, 50

[BCM+90] J.R. Burch, E.M. Clarke, K.L. Mcmillan, D.L. Dill, and L.J. Hwang. Symbolic model checking : 10^{20} states and beyond. In *Proceedings of 5th IEEE Symp. on logic and computer science*, 1990. 10

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 1020 states and beyond. *Information and Computation*, 98(2):142 – 170, 1992. 21

[BCOQ92] F. Baccelli, G. Cohen, GJ. Olsder, and JP. Quadrat. *Synchronization and Linearity: An algebra for discrete event systems*. John Wiley & Sons Inc., 1992. 18, 158

[BD91] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Transactions on Software Engineering*, 17:259–273, Mar 1991. 16, 157

[Ben00] A. Benasser. *L'accessibilité dans les réseaux de Petri : une approche basée sur la programmation par contraintes*. PhD thesis, Université des Sciences et Technologies de Lille, 2000. 8, 22, 23

[Ber86] G. Berthelot. Transformations and decompositions of nets. In Brauer, W., Reisig, W., and Rozenberg, G., editors, *Advances in Petri Nets 1986 Part I, Proceedings of an Advanced Course*, volume 254, pages 359–376. Springer-Verlag, 1986. NewsletterInfo: 27. 13

[Ber98] P. Berruet. *Contribution au recouvrement des systemes flexible de production manufacturiere: analyse de la tolerance et reconfiguration*. PhD thesis, Université des Sciences et Technologies de Lille, 1998. 19, 97, 98, 137

[BH06] T. Bourdeaud'huy and S. Hanafi. Scheduling of flexible manufacturing systems using timed petri nets and mathematical programming. *Proceedings of the 8th International Workshop on Discrete Event Systems*, 3:94–99, 2006. 87, 88

[BHY04] T. Bourdeaud'huy, S. Hanafi, and P. Yim. Efficient reachability analysis of bounded Petri netts using constraint programming. In

SMC'04, International Conference on Systems, Man and Cybernetics, La Hague, Hollande,, 2004. 54, 134, 172

[Bou04] T. Bourdeaud'Huy. *Techniques d'abstraction pour l'analyse et la synthése de réseaux de petri.* PhD thesis, Ecole Centrale de Lille, 2004. 23

[BRV04] B. Berthomieu, P.O. Ribet, and F. Vernadat. The tool TINA âĂŞ Construction of abstract state spaces for petri nets and time petri nets. *International Journal of Production Research*, 42(14):2741–2756, July 2004. 7

[BT09] T. Bourdeaud'huy and A. Toguyeni. Evaluation of mathematical programming models for the reconfiguration of reconfigurable manufacturing systems. In *Problems in Manufacturing*, pages 1941–1946, 2009. 19, 105

[CC88] J. Carlier and P. Chretienne. Timed Petri net schedules. *Advances in Petri Nets 1988*, 340:62–84, 1988. 17, 27, 28, 31, 33, 36, 54, 132, 157

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999. 2

[Chr84] P. Chretienne. Exécutions contrôlées des réseaux de Petri temporisés. *TSI. Technique et science informatiques*, 3(1):23–31, 1984. 32, 152, 155

[CL08] C.G. Cassandras and S. Lafortune. *Introduction to discrete event systems*, volume 11. Kluwer Academic Publishers, 2008. 1, 4

[Cla08] E. Clarke. The birth of model checking. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 1–26. Springer Berlin Heidelberg, 2008. 20

[CMS99] A. Cerone and A. Maggiolo-Schettini. Time-based expressivity of time Petri nets for system specification. *Theoretical Computer Science*, 216(1-2):1–53, March 1999. 5

[CPJ80] P. Chrétienne, Y. Pesqueux, and Grandjean J.C. *Algorithmes et pratique de programmation linéaire*. Technip, 1980. 12

[DA92] R. David and H. Alla. *Petri nets and grafcet: tools for Modelling discrete event systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992. 17, 157

[DA94] R. David and H. Alla. Petri nets for modeling of dynamic systems: a survey. *Automatica*, 30(2):175 – 202, 1994. 5

[Dea01] M. Diaz et al. *Les réseaux de petri - Modèles fondamentaux*. Hermes Science, Traité IC2 Information-Commande-Communication, 2001. 8, 9

[Dea03] M. Diaz et al. *Vérification et mise en œuvre des réseaux de Petri*. Hermes Science, Traité IC2 Information-Commande-Communication, 2003. 8

[DLBP05] F. De Lamotte, P. Berruet, and J.L. Philippe. Using model transformation for the analysis of the architecture of a reconfigurable system. In *IMACS'05 world congress*, Paris (France), july 2005. 19

[DTDC00] N. Dangoumau, A. Toguyeni, M. Dupas, and E. Craye. Reconfiguration process for automated production systems. In *MCPL'00*, Grenoble (France), july 2000. 19

[DYKG04] O.B. Driss, P. Yim, O. Korbaa, and K. Ghedira. Reachability search in timed petri nets using constraint programming. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 5, pages 4923–4928 vol.5, 2004. 17, 84, 158

[Fin93] A. Finkel. The minimal coverability graph for petri nets. *Advances in Petri Nets 1993, lecture notes in Computer Science*, 674:210–243, 1993. 13

[FJJM92] JC. Fernandez, C. Jard, T. Jéron, and L. Mounier. "on the fly" verification of finite transition systems. *Formal Methods in System Design*, 1992. 11

[God96] P. Godefroid. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032. Springer-Verlag Inc., New York, NY, USA, 1996. 14

[GP10] M. Gendreau and JY. Potvin. *Handbook of meta-heuristics*. Springer, 2010. 65

[GPG04] D. Gouyon, JF. Pétin, and Morel G. Control synthesis for product-driven automation. In *WODES'04*, september 2004. 19

[HBT07] S. Hadhri, T. Bourdeaud'huy, and A. Toguyeni. A mathematical programming approach for the reconfiguration of reconfigurable manufacturing systems with token identification. In *2007 IEEE International Conference on Systems, Man and Cybernetics*, pages 1351–1356. Ieee, October 2007. 96, 105, 106, 107, 112, 169

[HBYT12a] Y. Huang, T. Bourdeaud'huy, PA. Yvars, and A. Toguyeni. A constraint programming approach for generating firing sequences in timed Petri nets with token identification. In *11th International Workshop on Discrete Event Systems*, pages 149–156, 2012. 114, 133, 171

[HBYT12b] Y. Huang, T. Bourdeaud'huy, PA. Yvars, and A. Toguyeni. A constraint programming model for solving reachability problems in timed Petri nets. In *9ème Conférence Internationale de Modélisation, Optimisation et SIMulation*, pages 140–149, 2012. 89, 171

[HBYT12c] Y. Huang, T. Bourdeaud'huy, PA. Yvars, and A. Toguyeni. Using constraint programming for solving the reachability problem in timed Petri nets: evaluation of basic labeling strategies. In *14th IFAC Symposium on Information Control Problems in Manufacturing*, volume 14, pages 260–266, 2012. 78, 171

[HBYT13] Y. Huang, T. Bourdeaud'huy, P. Yvars, and Armand. Toguyeni. Approches incrémentales pour les réseaux de Petri temporisés fondées sur la programmation par contraintes.pdf. *Journal Européen des Systèmes Automatisés*, 27:77–92, 2013. 48

BIBLIOGRAPHY

[HDZJ04] S. Henry, E. Deschamps, E. Zamai, and M. Jacomino. Control law synthesis algorithm for discrete-event systems. In *proceedings of MCPL'04*, Santiago (Chili), 2004. 19

[He12] F. He. *Effective integrations of constraint programming, integer programming and local search for two combinatorial optimisation problems*. PhD thesis, University of Nottingham, 2012. 59

[HJJJ85] P. Huber, A.M. Jensen, L.O. Jepsen, and K. Jensen. Towards reachability trees for high-level petri nets. *Lecture Notes in Computer Science: Advances in Petri Nets 1984*, 188:215–233, 1985. NewsletterInfo: 27. 15

[HZ07] B. Hrúz and M. Zhou. *Modeling and control of discrete-event dynamic systems: With petri nets and other tools*. Springer, 2007. 4, 5

[IBM10] IBM. Ibm ilog solver v6.8 user's manual, 2010. 66, 79, 84, 167, 168

[Jen92] K. Jensen. Coloured petri nets - basic concepts, analysis methods and practical use. In *EATCS Monographs on Theoretical Computer Science*, volume 1, pages 1–234. Springer Verlag, 1992. 5

[JK91] R. Janicky and M. Koutny. Optimal simulations, nets and reachability graphs. In *Advances in Petri Nets, Lecture Notes In Computer Science*, volume 524, pages 205–226, 1991. 39, 40, 158

[KHJ+99] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. Van Brussel. Reconfigurable manufacturing systems. *Annals of the CIRP*, 48(32):527–540, 1999. 19

[KJ87] F. Krukeberg and M. Jaxy. Mathematical methods for calculating invariants in petri nets. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1987*, volume 266 of *Lecture Notes in Computer Science*, pages 104–131. Springer Berlin / Heidelberg, 1987. 72, 166

[KM69] R.M. Karp and R.E. Miller. Parallel program schemata. *Journal of Computer and Systems Sciences*, 3(2):147–195, mai 1969. 13

[Kos82] S.R. Kosaraju. Decidability of reachability in vector addition systems. In *Proc. of the 14th Annual ACM Symp. on Theory of Computing*, pages 267–281, San Francisco, 1982. 8

[Lau87] K. Lautenbach. Linear algebraic techniques for place/transition nets. In Brauer, W., Reisig, W., and Rozenberg, G., editors, *Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course*, volume 254, pages 142–167. Springer-Verlag, 1987. NewsletterInfo: 27. 12

[Lip76] R. Lipton. The reachability problem requires exponential space. Technical Report 62, New Haven, Connecticut: Yale University, Department of Computer Science, Research, Computer Science Dept., Yale University, 1976. 8

[LM89] J.B. Lassere and P. Mahey. Using linear programming in petri nets. *Operations Research*, 23(1):43–50, 1989. 12

[LNC+08] S. Lai, D. Nessi, M.P. Cabasino, A. Giua, and C. Seatzu. A comparison between two diagnostic tools based on automata and petri nets. In *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pages 144–149, 2008. 6

[Loi94] C. Loiseaux. *Vérification symbolique de programmes réactifs à l'aide d'abstractions.* PhD thesis, Université Joseph Fourier, Grenoble, 1994. 14

[MF76] P. Merlin and D. Farber. Recoverability of communication protocols implications of a theoretical study. *Communications, IEEE Transactions on*, 24(9):1036–1043, 1976. 5

[Mic07] A. Michalak. *Constraint programming for stochastic problems.* PhD thesis, AGH University of Science and Technology, 2007. 59

[MKI07] R. Maeno, M. Konishi, and J. Imai. Decomposition of time petri nets for solving optimal firing sequence problem. *Mem Fac Eng Okayama Univ (CD-ROM ...*, 41:44–51, 2007. 4, 5

[MKMH86] T. Murata, N. Komoda, K. Matsumoto, and K. Haruna. Petri Net-based controller for flexible and maintainable sequence control and its applications. *Industrial Electronics, IEEE Transactions on*, IE-33(1):1–8, 1986. 5

[Mur89] T. Murata. Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989. 34, 152, 154

[MV98] F. Michel and F. Vernadat. Maîtrise de l'explosion combinatoire. réduction du graphe de comportement. *RAIRO Technique et Science Informatiques*, 17:805–837, 1998. 4

[PBTZ05] J.F. Petin, P. Berruet, A. Toguyeni, and E. Zamai. Impact of information and communication emerging technologies in automation engineering: outline of the INTICA project. In *NeCST'05*, Ajaccio (France), october 2005. 97

[Pet62] C.A. Petri. Fundamentals of a theory of asynchronous information flow. In *IFIP Congress*, pages 386–390. North-Holland, 1962. 5

[Ram73] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, Massachusetts Institute of Technology, 1973. 5

[Ram74] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, Massachusetts Institute of Technology, 1974. 32, 33, 152, 154

[RBW06] F. Rossi, P.V. Beek, and T. Walsh. *Handbook of constraint programming (Foundations of artificial intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006. 26, 59, 61, 62, 65, 135

[Ric00] P. Richard. Modelling integer linear programs with petri nets. *RAIRO/Operations Research*, 34:305–312, 2000. 16, 38, 157

[RSET04] L. Recalde, M. Silva, J. Ezpeleta, and E. Teruel. Petri nets and manufacturing systems: an examples-driven tour. *Lectures on Concurrency and Petri Nets*, pages 71–89, 2004. 150

[SF12] J. Silva and P. Foyo. Timed petri nets. In *Petri Nets - Manufacturing and Computer Science*, pages 359–378. InTech, 2012. 5

[STC98] M. Silva, E. Teruel, and J.M. Colom. Linear algebraic and linear programming techniques for the analysis of place/transition net systems. In *LNCS 1492*, pages 309–37. Springer, 1998. 12

[TBC03] A. Toguyeni, P. Berruet, and E. Craye. Models and algorithms for failure diagnosis and recovery in FMSs. *International Journal of Flexible Manufacturing Systems*, 15:57–85, 2003. 19

[Tog04] A. Toguyeni. Reconfigurability analyser for automated production systems. In *INCOM'04*, Salvador-Bahia (Brazil), april 2004. 19

[Val91] A. Valmari. Stubborn sets for reduced state space generation. *Lecture Notes in Computer Science; Advances in Petri Nets 1990*, 483:491–515, 1991. NewsletterInfo: 33,39. 15

[Val98] A. Valmari. The state explosion problem. *Lectures on Petri Nets I: Basic Models*, 1491:429–528, 1998. 6

[VAM96] F. Vernadat, P. Azéma, and P. Michel. Covering steps graphs. In Springer-Verlag, editor, *17 th Int. Conf on Application and Theory of Petri Nets 96*, volume LNCS 1091, Osaka - Japan, 1996. 15

[Ver01] F. Vernadat. Contribution à la modélisation et à la vérification des systèmes communicants. Habilitation à Diriger des Recherches. LAAS/CNRS, mai 2001. 2, 8, 9

[Wan98] Jiacun Wang. *Timed Petri Nets, Theory and Application*. Kluwer Academic Publishers, 1998. 5

[XZ98] H.H. Xiong and M. Zhou. Scheduling of semiconductor test facility via petri nets and hybrid heuristic search. *Semiconductor Manufacturing, IEEE Transactions on*, 11(3):384–393, 1998. 5

[Yim00] P. Yim. Réseaux de petri, logique et théorie des ensembles : contributions à l'étude des systèmes dynamiques discrets. Habilitation à diriger des recherches. Université de Lille 1, Decembre 2000. 22

[YW94] M. Yamauchi and T. Watanabe. An approximation algorithm for the legal firing sequence problem of Petri Nets. *Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS '94*, 6:181–184, 1994. 6

[ZV99] M. Zhou and K. Venkatesh. *Modeling, simulation, and control of flexible manufacturing systems: a Petri net approach*. World Scientific, 1999. 150

Titre : Une approche incrémentale pour l'extraction de séquences de franchissement dans un Réseau de Petri Temporisé : application à la reconfiguration des systèmes de production flexibles

Cette thèse a pour objectif la génération de séquences de franchissement dans les Réseaux de Petri Temporisés (RdPT) en utilisant une approche incrémentale. Le verrou principal auquel est confronté ce travail est l'explosion combinatoire qui résulte de la construction classique du graphe d'accessibilité du RdPT. Nous proposons d'utiliser la notion de séquence de steps temporisés, afin d'exprimer progressivement l'ensemble des séquences de franchissements permettant de passer d'un état courant à un état cible. La notion de step temporisé correspond à une abstraction logique du comportement du système considéré. Le caractère incrémental de l'approche a pour objectif de gagner en efficacité. En effet, il consiste à exprimer tout nouvel état de la résolution par rapport à une profondeur K+1, en fonction d'un état atteint à la profondeur K. Ainsi, nous proposons plusieurs algorithmes de recherche incrémentale permettant d'améliorer l'efficacité de la résolution des problèmes d'accessibilité. Nous utilisons ensuite la programmation par contraintes pour modéliser le problème de recherche d'accessibilité dans un RdPT et mettre en œuvre notre approche incrémentale. Notre approche permet également d'ajouter des contraintes spécifiques à un contexte de résolution. Nous avons notamment utilisé cette possibilité pour proposer des techniques d'identification des jetons dans un RdPT borné, dans le cadre de la reconfiguration des systèmes manufacturiers. Nous concluons par l'évaluation de différentes applications constituant des benchmarks permettant d'illustrer l'efficacité des approches proposées.

Mots clés : Réseau des Petri Temporisé, Optimisation Combinatoire, Approche Incrémentale, Séquences des Franchissement, Problèmes d'Accessibilité, Programmation par Contraintes.

Title: An incremental approach for the extraction of firing sequences in Timed Petri Nets: application to the reconfiguration of flexible manufacturing systems

This PhD thesis is dedicated to the generation of firing sequences in Timed Petri Net (TPN) using an incremental approach. To reduce the complexity of the well-known problem âĂŞ combinatorial explosion issue, a unique sequence of timed steps is introduced to incrementally implicit traverse the underlying reachability graph of TPN, without needing its whole construction. This sequence of timed steps is developed based on the logical abstraction technique. The advantage of the incremental approach is that it can express any state just from the last step information, instead of representing all states before. Several incremental search algorithms are introduced to improve the efficiency of solving reachability problems. Constraint programming (CP) techniques are used to model and solve our incremental model, in which search strategies are developed to search for solutions more efficient. Our method can easily add specific constraints in realistic systems. Token identification techniques are developed to handle token confusion issues that appear when addressing the reconfiguration of manufacturing systems. Finally, experimental benchmarks are given to illustrate the effectiveness of approaches proposed in this thesis.

Keywords: Timed Petri Nets, Combinatorial Optimization, Incremental Approach, Firing sequences, Reachability Problem, Constraint Programming.