Fault Identification in Petri Nets: Theory and Implementation

Anastasia Ioannou
PhD Thesis

December 23, 2020

Fault Identification in Petri Nets: Theory and Implementation

Anastasia Ioannou

Abstract

In this thesis, we address the problem of fault identification in Petri nets: given a known Petri net and an unknown fault introduced to it, in terms of an additional (possibly unobservable) transition, we identify the fault, by only observing the executions of the system. We do this by translating the fault identification into a system of Integer Linear (In)Equations, which enable us to solve the problem using Integer Linear Programming. We prove its soundness, i.e., that the identified Petri net can indeed generate all the observed traces. Additionally, we observe and correct some inaccuracies in the existing literature in fault identification. We propose an approach to exploit the recent developments in the tools for checking satisfiability modulo theories. Through this approach, we empirically examine our research question, whether our fault identification technique is applicable in identifying faults in a number of examples. We have implemented the approach into a program that automatically codes the fault identification method into the input language of the Z3 SMT Solver. An evaluation of the applicability of our tool has been made by using a number of examples.

Dedicated to a special friend of mine, George Panayi, who was diagnosed with cancer and passed away few months before my submission.

Acknowledgements

I would like to express my gratitude to my initial supervisor, Rick M. Thomas, who had dedicated to me a lot of time and guided me during my PhD, even after his retirement. He was an inspiration to me, he supported me in every single aspect and without him it would have been difficult to complete such a work. He has taught me the methodology to carry out the research and to present the research works as clearly as possible.

I would also like to express my appreciation and personally thank my next supervisor, Mohammad Reza Mousavi, who took over Rick and who has supported me through my final year. His guidance was valuable to me and without his knowledge it would have been difficult to present such a work. His dedication to present a solid work was a motivation for me and I cannot thank him enough of what he managed in only one year of supervision.

I feel lucky and it has been a privilege working with them, not only because of their knowledge and expertise, but also because they both are beautiful people.

Special thanks to Carlos Diego Nascimento Damasceno and Mohammad Reza Mousavi for their contribution in my thesis and in the paper we are planning to publish after my submission.

The thesis is the result of the work done together with these three people and I really appreciate their hard work.

I would like to thank my parents, Andreas and Marina, my twin sister Christia and my younger brothers, Stelios and Christos, who have strongly believed in me, supported me and encouraged me during my PhD. I am extremely grateful to my parents for the love, caring and sacrifices for funding me, educating me and preparing me for my future.

I would also like to express my deep and sincere gratitude to my best friends, Maria O. and Maria C., for their constant love, support, patience, prayers and advice. Also, special thanks to Panagiotis V. for his love, pa-

tience and support, and to Joanna O. for her love, prayers, support and motivation. Without them, these years would have been stressful for me, but these people were always finding a way to motivate me to work harder, and whenever I was down they were always by my side helping me to stand on me feet again.

Special thanks to Andreas Poyias, who was a great marking partner in the modules we were teaching together, throughout these years.

Last of all, I would like to thank all of my friends and family who have believed in me and for their support and interest in my PhD area. I am so grateful having all these people in my life.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

For the past five decades, Petri nets have been extensively studied both in theory and application. They were first introduced in the early 1960s by Carl Adam Petri [49] for the description of chemical processes. Petri nets are mathematical modelling languages used for the description of distributed systems.

Nowadays, Petri nets are used for modelling, analysing and controlling discrete event systems, along with automata and other state-based formalisms. Although Petri nets have been extended in many directions and applied to several domains, such as manufacturing, transportation and communication, there still exist some open questions considering them. We will focus on one of these problems, namely identification of discrete event systems using Petri nets. The identification problem concerns constructing a formal model of an unknown system through observing its behaviour.

Several original approaches have been discussed considering the identification problem, among them is an approach by Hiraishi [35], who first introduced identification of safe Petri nets. There, an algorithm is proposed for constructing a Petri net model from a finite set of firing sequences. The algorithm consists of two phases. In the first phase, a language is identified in the form of a finite state automaton from the given firing sequences. In the second phase, the dependency relation is extracted from the language, and the structure of a Petri net is hypothesised. The algorithm is for a class of safe Petri nets, and its running time is bounded by a polynomial function in the size of inputs.

A system implementation may be faulty, for example, in that it may contain some additional transitions that are not present in the specification. In the context of this thesis, we call such additional transitions *faults*. These faults can lead to a *failure*, i.e., sequences that are not in the specification. In this thesis we aim to discover the additional transitions by observing the failures of the system. This is a well-known instance of the fault identification problem [30, 12, 23].

An approach [13], upon which the present thesis builds, deals with the fault identification problem within the framework of Petri nets without exploiting much knowledge about the nature of the fault. In particular, this approach identifies the structure of the faulty transitions of the system given its influence on the language of the faulty net. Then, the approach is generalized to deal with unobservable faults [13]. The generalized approach relies on

two important pieces of information: the structure of the fault-free specification, and the projection of the faulty system language on the set of non-faulty events.

Different approaches of the identification problem for free-labelled Petri nets have been proposed. Giua and Seatzu [30] deal with the problem of identifying a Petri net system, given the accepted strings of certain length. In particular, they consider the problem of identifying a free-labelled Petri net system, i.e., a net system where each transition is assigned a unique label. They show that the identification problem can be solved via a linear integer programming problem. They also discuss how additional structural constraints can be easily imposed on the net.

Similarly, Cabasino [12] and Dotoli [23] proposed an approach based on linear algebraic characterization of the net systems, given the accepted strings of certain maximal length. The set of transitions and the set of places are assumed to be known, while the net structure and the initial marking are computed by solving an integer programming problem. They also treat the problem of synthesizing a bounded net system starting from an automaton that generates its language. Finally, they show how the approach can also be generalized to the case of labelled Petri nets, where two or more transitions may share the same label. The output of their identification algorithm is always a deterministic net.

1.1.1 State of art for identification

Input data are usually given in terms of behavioral descriptions (e.g., transition system, language), and the identification (or synthesis) problem aims to address two main issues. First, decide whether for the given behavioral specification there exists a Petri net (of a given class) whose behavior coincides with the specified behavior. Secondly, provide a constructive procedure to determine the Petri net that satisfies the given specifications [14].

1.1.2 Theory of regions

A similar approach, that most contributions of identification topic are based on, is theory of regions [14]. Given a prefix-closed language L over some alphabet T, the theory of regions is trying to find a finite Petri net, which accepts L. The set of places for the Petri net is empty and the set of transitions consists of all possible transitions in T. The behavior of such Petri net

is not minimal, therefore, it needs to be minimized, such that the Petri net still accepts the language L, but it doesn't accept any words not in L. This can be achieved by adding places to the Petri net. The theory of regions provides a method to calculate the places, using regions. However, region-based techniques differ as there exist various types of Petri nets.

1.1.3 Synthesis problem

The basic synthesis problem consists of deciding whether a given finite initialized transition/net system is isomorphic to the reachability graph of some free labelled net system [2]. Ehrenfeucht and Rozenberg were the first who introduced the synthesis problem in order to model the sets of states that characterize marked places [24]. There exist several surveys regarding the synthesis of a Petri net. Among them are synthesis approaches introduced by Badouel and Darondeau [2], and by Lorenz [45].

Badouel and Darondeau [2] studied various types of net synthesis problems, depending on whether a given graph is isomorphic to the reachability graph or up to language equivalence, on whether synthesizing an arbitrary net system or an elementary net system, and on whether the net system is contact-free or not. For all types of synthesis problems, theoretical optimal solutions have been given. They then provide an algorithm which adapts to all synthesis problems, with one exception, and that runs in polynomial time taking into account the length of the alphabet of Petri nets with either sequential firing of with step firing rules and with respect to the number of places. Moreover, in [3, 4], the authors design explicit algorithms running in polynomial time and which are apply to most of the synthesis problems mentioned above.

Koutny et al. [41] study Petri nets with localities where transitions are partitioned into disjoint groups within which execution is synchronous and maximally concurrent. They generalise this type of nets by allowing each transition to belong to several localities. They also define this extension in a generic way for all classes of nets defined by net-types. They show that Petri nets with overlapping localities are an instance of the general model of nets with policies. They then construct nets with localities from behavioural specifications given in terms of finite step transition systems. To conclute, they outline some initial ideas concerning net synthesis when the association of transition to localities is not given and has to be determined by the synthesis algorithm.

Cortadella, Kishinevsky, Lavagno and Yakovlev [18], introduce a method which, given a finite state model(transition system), synthesizes a safe, place-irredundant PN with a reachability graph that is bisimilar to the original transition system.

Carmona et al. [16] address an efficient synthesis approach for concurrent systems. An algorithm for k-bounded Petri nets synthesis based on the theory of general regions is presented. A bounded Petri net is always provided in case it exists. Otherwise, they propose heuristically split events into multiple transitions, and the algorithm always guarantees a visualization object. A minimal Petri net with bisimilar behavior is obtained when the original transition systems is excitation closed from sets of states to multisets of states. Experimental results show a significant net reduction when compared with approaches for the synthesis of safe Petri nets.

In [44], the authors, given a set of scenarios, find an answer to the question whether this set equals the set of all executions of a Petri net. They define regions of partial languages, and they prove a characterization of partial languages of executions of Petri nets based on this notion of regions. Finally they show that this notion of regions is consistent with the notion of regions of trace languages.

In [43], the authors present, given a finite set of labeled partial orders representing a partial language, how to compute a (finite) marked Petri net with minimal set of runs, such that each specified labeled partial order is a run of the net. Finally, they presented, for the first time, an effective algorithm to test, whether the computed net has more runs than specified or not. This decides the synthesis problem, since the synthesis problem has a solution if and only if the computed net does not have more runs than specified.

Additionally, in [45], Lorenz et al. present a survey on methods for the synthesis of Petri nets from behavioral descriptions given as languages. They consider Petri nets, elementary Petri nets and Petri nets with inhibitor arcs. For each net class they consider classical languages, step languages and partial languages as behavioral description. All methods are based on the notion of regions of languages. Altogether, they present a framework for region-based synthesis of Petri nets from languages which integrates almost all known approaches by several new algorithms that involve integer programming methods. It was the first synthesis algorithms for step languages that had ever been introduced.

The [6] by Bergenthum et al., tackles synthesis of Petri nets from infinite

partial languages. More precisely, they introduce terms built from labeled partial orders (LPOs) and composition operators including iteration. Moreover, they consider operators for sequential and parallel composition as well as a union operator. Given a term constructed this way from a finite set of LPOs, they show how to synthesize a finite Petri net from this term such that the behaviour of the net coincides with the set of LPOs represented by the term – if such a net exists. The synthesis approach is based on the so called theory of regions. The synthesized Petri net has minimal net behaviour including the behaviour specified by the given term.

1.1.4 Process mining

An actual application area of synthesis methods is the area of process mining [55, 6]. The goal of process mining is to automatically generate a process model from an event log. In other words, process mining techniques aim to discover, monitor and improve real processes by extracting knowledge from event logs.

Process mining is quite different from constructing a Petri net on the basis of regions because the notion of completeness is much weaker than the typical assumption when using regions (i.e., a complete transition system). In synthesis one assumes a complete description of the system, only a partial description of the system is assumed in process mining. For process mining one assumes that the log contains only a fraction of the possible behavior. Existing process mining techniques have problems dealing with large event logs referring to many different activities [59]. During process mining, specialized data mining algorithms are applied to event log data in order to identify trends, patterns and details contained in event logs recorded by an information system.

In [59], van der Aalst et al. propose a generic approach to decompose process mining problems. The decomposition approach is generic and can be combined with different existing process discovery and conformance checking techniques. It is also noted that is possible to split computationally challenging process mining problems into many smaller problems that can be analyzed easily and whose results can be combined into solutions for the original problems.

In [58], van der Aalst and Weijters propose a technique for process mining. This technique uses workflow logs to discover the workflow process as it is actually being executed. The process mining technique proposed in the paper

can deal with noise and can also be used to validate workflow processes by uncovering and measuring the discrepancies between prescriptive models and actual process executions.

In [17], Carmona et al. present Genet, a tool that allows the derivation of a general Petri net from a state-based representation of a system. The tool supports two modes of operation: synthesis and mining. Moreover, in [15], Carmona et al. present a new method for the synthesis of Petri nets from event logs in the area of process mining. The method derives a bounded Petri net that over-approximates the behavior of an event log. The most important property is that it produces a net with the smallest behavior that still contains the behavior of the event log. The methods described in this paper have been implemented in a tool and tested on a set of examples.

In [57], the authors developed techniques for discovering workflow models. The starting point for such techniques is a so-called "workflow log" containing information about the workflow process as it is actually being executed. They present a new algorithm to extract a process model from such a log and represent it in terms of a Petri net. However, they demonstrate that it is not possible to discover arbitrary workflow processes. In the paper, they explore a class of workflow processes that can be discovered and they show that the algorithm can successfully mine any workflow process.

1.2 Problem statement

In all of the above-mentioned techniques the aim is to construct a Petri net from a set of observations or specifications. My thesis also treats a related problem, but with a specific set of hypotheses: I assume that I have access to an original specification of system in terms of a Petri net and also finite traces of its faulty implementation (up to a given length). Faults in this setting are additional transitions that were not present in the specification and were introduced due to a mistake (e.g., manufacturing fault, or programmer's error) in the implementation. I would like to construct (synthesise, mine) a Petri net from the implementation traces with the aim of identifying the implemented fault. In this setting, fault identification amounts to identifying the (weighted) arcs connecting the faulty transitions to the original specification traces. I call this problem the *fault identification problem* in Petri nets.

In this thesis, I address fault identification in Petri nets in two different settings: the first setting, which serves as an exercise to set the scene for my

solution, assumes that the occurrences of faulty transitions in the traces of the implementation are observable. In other words, our first research question is as follows:

> **RQ1** Is it possible to efficiently identify observable faults in a system built from a known Petri net.

The second research question, which is closer to the real setting, removes the assumption about the observability of the faulty transitions. My second research question is hence phrased as follows:

> **RQ2** Is it possible to identify unobservable faults in a system built from a known Petri net.

I answer both research questions by translating the fault identification into a system of Integer Linear (In)Equations, which enable us to solve the problem using Integer Linear Programming. I prove its soundness, i.e., that the identified Petri net can indeed generate all the observed traces. Subsequently, I answer the followig research question by implementing my approach, designing a benchmark of examples of various degrees of complexity, and evaluating the efficiency of fault identification on the designed benchmark:

> **(RQ3)** How scalable is my fault identification technique for unobservable faults?

1.3 Contributions of the thesis

In order to answer this question, we push the state of the art:

- correcting the inaccuracies in the existing formulation fault identification problem [12] corresponding to **RQ2**,
- implementing my approach using a state of the art SMT solver. Through this implementation, we empirically examine the research question, whether my fault identification technique is efficient in identifying the fault,

- implementing the approach into a program that automatically codes the fault identification method into the input language of the Z3 SMT Solver, and
- designing a benchmark and evaluating the applicability of my tool using the designed benchmark, addressing **RQ3**.

1.4 Structure of the thesis

The rest of this thesis is organised as follows. In Chapter 2, I provide an overview of the formal definitions concerning Petri nets and faults and formalise our problem statement. In Chapter 3, I rephrase a solution scheme in our setting, which is mainly due to Cabasino [12]. In this solution scheme, the problem is translated into a system of linear inequations, assuming that faults are observable, pertaining **RQ1**. Subsequently, in Chapter 4, I drop the assumption that the faulty transition is observable, addressing **RQ2**. In Chapter 5, I review the approach of Cabasino [14], which has inspired our approach and report the issues we observed. In Chapter 6, I present my implementation, which translates the specification of the original Petri net and my systems of linear (in)equations into the input language of the Z3 SMT Solver. In Chapter 7, I design a benchmark comprising a number of examples of various sizes, and I evaluate my proposed approach and the implementation on the designed benchmark, addressing **RQ3**. Finally, I conclude the thesis and present the directions of my future research in Chapter 8.

Chapter 2

Preliminaries

2.1 Modelling using Petri nets

Model-based analysis is the study of using models in order to express, construct, and analyse the architecture of systems. Different notations and formalisms have been proposed for system modelling at different levels of abstraction. Petri nets is one such formalism that has been introduced for modelling and analysis of concurrent systems at a high level of abstraction.

In a Petri net, places identify the state (conditions, configurations) of the system being modelled (e.g., idle, working, queuing, failed). In each place, there can be zero or more tokens representing the valuation of that state at each moment. Different events occurring in the system (e.g., end of a task, repair, failure) are represented by transitions. In order for an event to occur, its source and target states conditions must be satisfied. This is indicated by input arcs; conditions are then denoted by the presence of tokens in the source of their input arcs. Output arcs represent the resulting states of event occurrences. As a result of firing a transition, i.e., occurrence of an event, tokens are placed at the target places of the output arcs. The number of tokens in a place represent the valuation of the state; for instance, in a data processing system, where transitions represent the data processing events and tokens in source and target places represent input and output data, respectively. Systems containing conditions and events are known as Condition/Event-systems [8, 61].

2.2 Languages: basic definitions

In this section we will introduce some basic definitions concerning alphabets and languages.

Definition 1. An *alphabet* $\mathcal{A}$ is a non-empty finite set of symbols.

We let $\mathcal{A}^*$ denote the set of all finite strings or words, including the empty string λformed by concatenating zero or more symbols from alphabet $\mathcal{A}$. $\mathcal{A}^+$ denotes the set of all non-empty finite sequences of elements of $\mathcal{A}$. □

For example, consider alphabet $\mathcal{A} = \{a, b\}$. Then, according to the above-given definition, we have

$$\mathcal{A}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, aba, baa, \dots\}$$

This is well-known as the *Kleene star* operator.

Definition 2. A *language* L over alphabet $\mathcal{A}$ is a set of words over the alphabet, i.e., $L \subseteq \mathcal{A}^*$.

For a $q \in \mathbb{N}$ we let $L_{\leq q}$ denote the set of all sequences in L of length at most q. □

Considering alphabet $\mathcal{A} = \{a, b\}$, examples of languages $\mathcal{A}$ include $\emptyset$ (the empty language), $\{a, b\}$ ($\mathcal{A}$ itself), $\{\lambda, a, b, aa, ab, ba, bb\}$ (all words of size at most two) and $\mathcal{A}^*$.

Definition 3. The string α is a *prefix* of a string s, if there exists a (possibly empty) string β, such that

$$s = \alpha\beta$$

□

In other words, α is a prefix of a string s, if α is a substring of s appearing at the beginning of s. For example if $s = aba$, we have that ab is a prefix of s.

Definition 4. A language L over $\mathcal{A}$ is said to be *prefix-closed* if whenever $\alpha \in L$ and β is a prefix of α, then $\beta \in L$, i.e.:

$$\alpha \in L, \beta \text{ a prefix of } \alpha \Longrightarrow \beta \in L.$$

□

Definition 5. Let $L \subseteq \mathcal{A}^*$ then the set of prefixes of L, denoted by $Pref(L)$, is defined as follows:

$$Pref(L) = \{\sigma \in \mathcal{A}^* : \exists t \in \mathcal{A}^*, \sigma t \in L\}$$

□

In general $L \subseteq Pref(L)$, where $Pref(L)$ consists of all the prefixes of all strings in L. For, example, if $L = \{ab\}$ then $Pref(L) = \{\epsilon, a, ab\}$. If L is prefix-closed then $L = Pref(L)$.

2.3 Petri nets: basic definitions

Petri nets provide a formal method for modelling and analysing a large variety of systems; other alternative methods include automata, process algebras, and many other state-based formalisms. The pictorial representation of a Petri net is a graph that consists of two kinds of nodes; circles called *places* and bars or boxes called *transitions*. Places and transitions are connected by arcs, either from a place to a transition or vice-versa. There are no arcs between two places, nor between two transitions. Input arcs connect a transition to a place and output arcs connect a place to a transition. They represent the pre-conditions and the post-conditions of transitions respectively. Input and output arcs are labelled with weights. The weight of an input arc indicates how many tokens must be put in a place after firing a transition, whereas the weight of the output arc indicates how many tokens must be consumed from a place when a transition is fired. In simple words, the weight of an arc denotes how many tokens must be removed from a place and how many tokens must be put to another place when a transition is fired.

After this intuitive explanation of the structure of Petri nets, their formal definition is given below.

Definition 6. A Petri net is a 4-tuple (P, T, F, ω) where:

- P is a finite set of places
- T is a finite set of transitions
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation of the net
- $\omega : F \to \mathbb{N}$ is the weight function

□

Places may contain markers called *tokens* which control the execution of the Petri net. A place can have a non-negative number of tokens. The *initial marking* is the marking of the Petri net at the beginning. Petri nets with such initial markings are called *marked* Petri nets.

Definition 7. Let $N = (P, T, F, \omega)$ be a Petri net. A marking of N is a function $\mu : P \to \mathbb{N}$. We denote the initial marking by μ_0. By $\mu(p)$ we denote the number of tokens in place p in a marking μ. □

As mentioned above, pre and post conditions are the input and the output arcs of places and transitions. Transitions consume tokens from the input place and produce tokens in the output place. The distribution of the tokens in places results in a new marking. The following definition provides an explicit notation for pre- and post-conditions in terms of arc weights.

Definition 8. Consider a Petri net, $N = (P, T, F, \omega)$; Let $Pre : P \times T \to \mathbb{N}$ and $Post : T \times P \to \mathbb{N}$ be defined as follows:

$$\begin{aligned} Pre(p,\ t) &= \omega(p, t) \qquad \forall p \in P, t \in T \\ Post(p,\ t) &= \omega(t, p) \qquad \forall p \in P, t \in T \end{aligned}$$

If $(p, t) \notin F$ then we define $Pre(p, t) = 0$ and similarly, if $(t, p) \notin F$ then $Post(t, p) = 0$.

We often assume an arbitrary, yet fixed, ordering of places $|P| = [p_1, \cdots, p_n]$; in this context, $Pre(t)$ and $Post(t)$ are the vectors defined as follows:

$$Pre(t) = \begin{bmatrix} Pre(p_1, t) \\ \vdots \\ Pre(p_n, t) \end{bmatrix}$$

and

$$Post(t) = \begin{bmatrix} Post(p_1, t) \\ \vdots \\ Post(p_n, t) \end{bmatrix}$$

Similarly, we also fix an ordering on transitions $|T| = [t_1, \cdots, t_m]$; in this context, we define

$$Pre = [Pre(t_1), \ldots, Pre(t_m)]$$

and

$$Post = [Post(t_1), \ldots, Post(t_m)]$$

Note that Pre and $Post$ are both $n \times m = |P \times T|$ matrices. Also C, is called the incidence matrix and is defined as $Post - Pre$ is the incidence matrix.

If σ is a non-empty sequence $(v_1, v_2, \ldots, v_k)$ of transitions, then we define:

$$Pre(\sigma) = Pre(v_1) + Pre(v_2) + \ldots + Pre(v_k)$$

,

$$Post(\sigma) = Post(v_1) + Post(v_2) + \ldots + Post(v_k)$$

. □

Having presented the definition of (marked) Petri nets, we now present the execution semantics of Petri nets based on some rules.

A transition t is *enabled* at a marking μ if all of its input places (places which are connected to transition t by some input arcs) have at least as many tokens in μ as the weight of the arc from that place to t. The transition t then may fire by consuming the specified number of tokens from the pre-condition places and then generating the specified number of tokens to each of the post-condition places of the transition. We call this the *firing* of a transition.

A transition t is said to be *loop-free*, if there is no place p that is both an input and an output place of this transition.

Definition 9. A transition t is *enabled* under a marking μ if $\mu(p) \geq Pre(p, t)$ for every $p \in P$; otherwise, t is said to be *disabled*.

Consider a transition t enabled at μ; after firing t at μ, a new marking μ' is generated defined by

$$\mu'(p) = \mu(p) + Post(p, t) - Pre(p, t). \tag{2.1}$$

We write $\mu \xrightarrow{t} \mu'$ to indicate that marking μ' is reachable from marking μ when transition t is fired.

Similarly, we write $\mu \xrightarrow{\sigma} \mu'$ to indicate that marking μ' is reachable from marking μ when a sequence of transitions $\sigma = t_1 \ldots t_n$ is fired. □

The following definition formalizes the concepts of firing sequences and firing vector; the former is merely a sequence of consecutively enabled transitions and the latter is the number of occurrences of each transition in such a sequence.

Definition 10. A sequence $\sigma = t_1 \dots t_n$ of transitions, where $t_1 \dots t_n \in T^*$, is said to be the *firing sequence* of a Petri net if it is enabled at μ_0. For example, transition t_1 is enabled at μ_0, then arriving at marking μ_1, t_2 is then enabled at μ_1, and so on.

Given a firing sequence $\sigma \subseteq T^*$, the firing vector of σ, denoted by $\pi(\sigma)$, is the vector

$$\pi(\sigma) = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

where m is the number of transitions, and y_i is the number of times transition t_i appears in σ, for all $i = 1, \dots, m$. □

The vector μ_σ denotes the marking obtained from μ_0 after a sequence of transitions σ, i.e.: $\mu_0 \xrightarrow{\sigma} \mu_\sigma$. We say that μ_σ is undefined if σ is not enabled at μ_0. If σ is the empty sequence of transitions we say that σ is enabled at μ_0 and define μ_σ to be μ_0.

We define the *state equation* to be

$$\mu_\sigma = \mu_0 + (Post - Pre) \cdot \pi(\sigma)$$

The proof of this equation is by a straightforward induction on the total number of transitions in σ.

Definition 11. Consider a net with initial marking $\langle N, \mu_0 \rangle$, where $N = (P, T, F, \omega)$ is a Petri net. The *reachability set* $R(N, \mu_0)$ of N is then the set of all markings μ that are reachable from the initial marking μ_0 through some firing sequence σ from μ_0 to μ, i.e.:

$$R(N, \mu_0) = \{\mu : \mu = \mu_\sigma \; for \; some \; \mu \in T^*\}.$$

□

Note that $\mu_0 \in R(N, \mu_0)$ and that $\lambda \in L(N, \mu_0)$ by definition.

Definition 12. The *language* of a Petri net N from the initial marking μ_0, denoted by $L(N, \mu_0) \subseteq T^*$, is the set of all sequences of transition firings that are enabled from the initial state.

For all natural numbers $q \in \mathbb{N}$, the set $L_{\leq q}(N, \mu_0) \subseteq T^*$ is the set of all firing sequences of maximum length q in $L(N, \mu_0)$. □

Whenever the initial marking μ_0 is known from the context we will write $L(N)$, $L_{\leq q}(N)$ and $R(N)$ instead of $L(N, \mu_0)$, $L_{\leq q}(N, \mu_0)$ and $R(N, \mu_0)$, respectively.

2.4 Subclasses and extensions of Petri nets

Some properties of Petri nets are hard to analyse. For that reason, Petri nets have been divided into subclasses with lower complexity bounds for their analysis and they are defined by some restrictions on their structure. The following are some of the main subclasses of Petri nets [48].

- A *state machine* is a restricted Petri net where each transition has exactly one input and one output place. On the other hand, a place can have more than one input and output transitions, but only one of those transitions can fire at a time, as the marking always consists of a single token. Such transitions are said to be in *conflict*. State machines have less expressive / computational power than Petri nets in general.

- A *marked graph* is a Petri net where each place has exactly one input and one output transition. In marked graphs parallel activities are allowed. In other words, a transition can have more than one input/output places. This phenomenon can be used to model synchronisation.

- A *free-choice net* is a Petri net such that every arc from a place is either the only output arc of the place, or the only input arc to a transition.

To facilitate modelling more complex phenomena, Petri nets have also been extended with various features. The following is an example of such an extension.

- A *Petri net with inhibitor arcs* is a net with special arcs, called *inhibitor arcs*. An inhibitor arc from a place p to a transition t, allows t to fire if and only if p does not contain tokens in it. Graphically they are represented as arcs with a hollow circle instead of an arrowhead.

2.5 Representing faults in Petri nets

As discussed in Chapter 1, a system implementation may be faulty in that it may contain spurious transitions that are not present in the specification. These faults can lead to failures, i.e., traces that are not in the language of the specification. The subject matter of this Chapter (and much of the remainder of the thesis) is fault identification. In other words, we aim to discover the spurious transition by scrutinizing the failures, i.e., the spurious traces. The following definition formalises this concept.

Definition 13. Consider a marked Petri net $\langle N, \mu_0 \rangle$, where $N = (P, T, F, \omega)$ and μ_0 is the initial marking; this Petri net is assumed to represent the fault-free specification. A faulty extension $\langle N^f, \mu_0 \rangle$, is a Petri net where $N^f = (P, T^f, F^f, \omega^f)$ and μ_0 is the initial marking, such that the following conditions hold:

- P is the set of places
- T^f is the set of transitions including faulty transitions, such that $T \subseteq T^f$
- $F^f \subseteq (P \times T^f) \cup (T^f \times P)$ is the flow relation of the net, such that $F \subseteq F^f$ and $F^f \cap ((P \times T) \cup (T \times P)) = F$
- $\omega^f : F^f \to \mathbb{N}$ is the weight function, satisfying the following constraints

$$\begin{aligned} \omega^f(p,t) &= \omega(p,t) \qquad \forall p \in P, t \in T \\ \omega^f(t,p) &= \omega(t,p) \qquad \forall p \in P, t \in T \end{aligned}$$

In other words, if we restrict the flow relation F^f of N^f to F and ignore the transitions in $T^f \setminus T$, we have our original Petri net N. □

As specified above, we assume that the transitions and places of the specification are also present in the faulty system. This indicates that the faulty system has the same underlying structure as the fault-free specification; however, the former includes some additional faulty transitions.

Our goal is to identify the fault of the system by using the language which is generated by the faulty Petri net. Specifically, this can be done by investigating the additional sequences that originate from the additional

faulty transitions. We assume that T^f is the extended set of (faulty and non-faulty) transitions leading to a faulty behaviour and we also assume that the faulty transitions are unobservable (note that there are no new places in the faulty implementation). The firing of the unobservable transitions cannot be detected directly. However, we scrutinise finite traces of a maximal length (called q below) and from them infer which extra faulty transitions are present.

In other words, we will be given a Petri net N with an initial marking and a language $\mathcal{L}$, which represents the set of firing sequences of the faulty extension N^f of N up to a certain length, and our goal is to deduce what N^f is from $\mathcal{L}$.

Definition 14. Consider a Petri net N and its faulty extension N^f; the projection of a singleton firing sequence of N^f into that of N, denoted by $\Downarrow_T (\_) : T^f \to T \cup \{\lambda\}$, is defined as follows:

$$\Downarrow_T (t) = \begin{cases} t & \text{if } t \in T \\ \lambda & \text{if } t \in T^f \setminus T \end{cases} \tag{2.2}$$

Additionally, consider σt_j, where σ is a sequence and t_j is a transition, then we have that:

$$\Downarrow_T (\sigma t_j) = \Downarrow_T (\sigma) \Downarrow_T (t_j) \quad \forall\ \sigma \in (T^f)^*, t_j \in T^f$$

□

We refer to $\Downarrow_T$ as the *projection* from $(T^f)^*$ onto T^*. We extend $\Downarrow_T$ to a language on $(T^f)^*$ as expected:

$$\Downarrow_T (L) = \{\Downarrow_T (\alpha) : \alpha \in L\} \text{ for , where } L \subseteq (T^f)^*$$

.

2.6 Problem formalization

2.6.1 Observable faults

We first formalise our research question concerning observable faults (**RQ1**):

Problem 15. Given a net N and a finite language $L^f \subseteq ((T^f)^*)^{\leq q}$ with a maximum length of q such that $L_{\leq q}(N, \mu_0) \subseteq L^f$. Decide whether there is a Petri net N^f extending N such that $L^f = L_{\leq q}(N^f)$ and, if the answer is positive, construct one such N^f. The unknowns here are F^f and ω^f. □

2.6.2 Unobservable faults

Next, we move the setting with unobservable faults, i.e., our second research question (**RQ2**). The following problem formalises our **RQ2**:

Problem 16. Given a net N and a finite language $L^f \subseteq (T^*)^{\leq q}$ with a maximum length of q such that $L_{\leq q}(N, \mu_0) \subseteq L^f$. Decide whether there is a faulty system N^f extending N such that $L^f = \Downarrow_T (\mathcal{L}_{\leq q}(N^f))$ and, if the answer is positive, construct one such N^f. The unknowns here are T^f, F^f and ω^f. □

The goal is to identify the structure of the faulty system, based on its observable language which is known. When faults are not observable themselves, they should be identified based on the other transitions they enable. (The unobservable transitions are also known as silent transitions [13].) In other words, scrutinising $L^f_{\leq q} \setminus L_{\leq q}$ is they key to fault identification.

Chapter 3

Observable faults

3.1 Problem setting

We start with the simple and well-studied setting for the fault identification problem, namely, when faults are observable (**RQ1**). The goal is to characterise the enabledness and disabledness of non-faulty (original) and faulty transitions into integer linear (in)equations by observing original and faulty sequences and hence, reduce the fault identification problem into a linear integer programming problem based on the aforementioned integer linear (in)equations. The assumption that the faults are observable may not be practical in all settings and in fact, we remove this assumption in the next chapters; however, in some settings one may be able to indirectly observe the presence of faulty transitions, e.g., by observing the execution time of various observable transitions. This is a well-studied problem and in this chapter, we mostly review the approach proposed by Cabasino [14] and do not claim much novelty about it. The main purpose of this chapter is to set the scene for the more inovlved setting concerning unobservable faults (**RQ2**), which is presented in the subsequent chapters.

As before, we assume an arbitrary yet fixed order of non-faulty transitions $[t_1, t_2, t_3, \ldots, t_n]$ and faulty transitions $[f_1, f_2, f_3, \ldots, f_k]$. Let q be a specified length of the language of the net system and $L_{\leq q}$, respectively $L^f_{\leq q}$, be the prefix-closed subsets of L, respectively L^f, that contain all strings of up to and including length q.

Definition 17. In order to characterise the different enabledness and disabledness conditions, we define the following four sets of strings:

$$PL(N)_{\leq q} = \{(\sigma, t_j) | \sigma \in L_{\leq q-1}, t_j \in T, \sigma t_j \in L_{\leq q}\} \quad (3.1)$$
$$PNL(N)_{\leq q} = \{(\sigma, t_j) | \sigma \in L_{\leq q-1}, t_j \in T, \sigma t_j \notin L_{\leq q}\} \quad (3.2)$$
$$PL(N^f)_{\leq q} = \{(\sigma, t_j) | \sigma \in L^f_{\leq q-1}, t_j \in T^f, \sigma t_j \in L^f_{\leq q} \setminus L_{\leq q}\} \quad (3.3)$$
$$PNL(N^f)_{\leq q} = \{(\sigma, t_j) | \sigma \in L^f_{\leq q-1}, t_j \in T^f, \sigma t_j \notin L^f\} \quad (3.4)$$

where (σ, t) is a pair of a sequence $\sigma \in (T^f)^*$ and a transition $t \in T^f$ in the net. □

3.2 Characterising non-faulty sequences

Consider a non-faulty sequence σ and a transition t_j such that $(\sigma, t_j) \in PL(N)_{\leq q}$. Then t_j must be enabled from the marking $\mu_0 + (Post - Pre) \cdot \pi(\sigma)$ and the following relation must hold

$$\mu_\sigma \geq Pre(t_j),$$

where μ_σ represents the marking after firing σ. The above equation can be rewritten as

$$\mu_0 + Post \cdot \pi(\sigma) - Pre \cdot [\pi(\sigma) + t_j] \geq \vec{0} \quad (3.5)$$

This equation forms the first constraint for characterising the non faulty sequences; it makes sure that all non-faulty transitions remain enabled in the faulty net as well.

3.3 Characterising faulty sequences

Consider now a sequence σ and a transition t_j such that $(\sigma, t_j) \in PNL(N^f)_{\leq q}$. It follows from the definition of PNL that σ is in the language of the faulty net and t_j is not enabled from the marking $\mu_\sigma + (Post - Pre) \cdot \pi(\sigma)$. Hence, there must exist at least one place p_i such that the following constraint holds:

$$\mu_\sigma(p_i) < Pre(p_i, t_j),$$

We define a vector

$$S_{\sigma,t_j} = \begin{bmatrix} s_1 \\ \vdots \\ s_m \end{bmatrix} \in \{0,1\}^m$$

where m is the number of places such that for each p_k, $1 \leq k \leq m$, s_k is defined as follows:

$$s_k = \begin{cases} 0 & \text{if } \mu_\sigma(p_k) < Pre(p_k, t_j) \\ 1 & \text{if } \mu_\sigma(p_k) \geq Pre(p_k, t_j) \end{cases} \tag{3.6}$$

We omit the subscripts σ and t_j, when they are clear from the context.

We also assume that $K = \max\mu_\sigma(p_k)$; hence, we have that $\mu_\sigma(p_k) \leq K$ and thus that $\mu_\sigma(p_k) - K \leq 0$.

For each place p_k, we aim to prove that $\mu_\sigma(p_k) - Pre(p_k, t_j) < K \cdot s_k$; to this end, we distinguish the following two cases based on the status of s_k:

$s_k = 0$ Then, we have that $\mu_\sigma(p_k) - Pre(p_k, t_j) < 0$, i.e., $\mu_\sigma(p_k) - Pre(p_k, t_j) < K \cdot s_k$ or

$s_k = 1$ Then, we have that $\mu_\sigma(p_k) - Pre(p_k, t_j) < K$, i.e., $\mu_\sigma(p_k) - Pre(p_k, t_j) < K \cdot s_k$.

From the above-mentioned statement and because we have that $\mu_\sigma = \mu_0 + (Post - Pre) \cdot \pi(\sigma)$, we obtain that:

$$\mu_0 + (Post - Pre) \cdot \pi(\sigma) - Pre(t_j) - K \cdot S < \vec{0}$$

Example 18. Consider a Petri net with 3 places such that transition t_1 is disabled at the end of the trace σ; we obtain the following equations:

$$\mu_\sigma(p_1) - Pre(p_1, t_1) < 0$$

$$\mu_\sigma(p_2) - Pre(p_2, t_1) < 0$$

$$\mu_\sigma(p_3) - Pre(p_3, t_1) < 0$$

At least one, but not all three, of these constraints must be satisfied. This restriction can be modelled by combining the technique just introduced with multiple-choice constraint as follows:

$$-Ks_1 + \mu_\sigma(p_1) - Pre(p_1, t_1) \leq -1$$

$$-Ks_2 + \mu_\sigma(p_2) - Pre(p_2, t_1) \leq -1$$

$$-Ks_3 + \mu_\sigma(p_3) - Pre(p_3, t_1) \leq -1$$

and

$$\sum_{k\leq 3} s_k < 3$$

.

The K and s_i's are chosen to indicate when the constraints are satisfied. The multiple-choice constraint $\sum_{k\leq 3} s_k < 3$ implies that at least one variable $s_k = 0$, so that, as required, at least one constraint must be satisfied. □

3.4 From Fault Identification to Linear Integer Programming

The following theorem [14] combines the previous results and provides a solution to **RQ1**, that is the fault identification problem with observable faults, by translating the problem into a system of linear algebraic inequations.

Theorem 19. *A net system $\langle N, \mu_0\rangle$ is a solution to the identification problem, Problem **RQ1**, if and only if the following set of linear algebraic constraints $G_m(PL(N)^f, PNL(N)^f)$ are satisfied [14]:*

$$\begin{cases} \mu_0 + Post \cdot \pi(\sigma) - Pre \cdot (\pi(\sigma) + \vec{t_j}) \geq \vec{0} & \forall(\sigma, t_j) \in PL(N)^f \\ -KS_{\sigma,j} + \mu_0 + Post \cdot \pi(\sigma) - Pre \cdot [\pi(\sigma) + \vec{t_j}] < \vec{0}_m & \forall(\sigma, t_j) \in PNL(N)^f \\ \vec{1}^T S_{\sigma,j} \leq m - 1 & \forall(\sigma, t_j) \in PNL(N)^f \\ \mu_0 \in \mathbb{N}^m \\ Pre, Post \in \mathbb{N}^{m\times n} \\ S_{\sigma,j} \in \{0, 1\} \end{cases} \tag{3.7}$$

where $y = \pi(\sigma)$ and $PL(N)^f, PNL(N)^f$ are given above.

In general, the solution of Theorem 19 is not unique; typically among such solutions, one is looking for the "smallest" one, which provides a concise explanation of the fault. In order to select one among these solutions the author of [14] introduces the following proposition, Proposition 20, where by choosing a given performance index and by solving an integer programming problem, determines a Petri net system that minimizes the performance index mentioned above. However, the solution may still not be unique.

Proposition 20. *Let* $g(Pre(f_1), \ldots, Pre(f_n), Post(f_1), \ldots, Post(f_n))$ *be defined as*

$$\sum_{i=1}^{m} \sum_{j=1}^{n} b_{i,j} Pre(p_i, f_j) + c_{i,j} Post(p_i, f_j) \tag{3.8}$$

where $b_{i,j}, c_{i,j} \in \mathbb{R}_0^+$. *In the rest of the thesis we choose all given coefficients* $b_{i,j}$ *and* $c_{i,j}$ *be equal to* 1. *So the above equation can be rewritten as:*

$$\begin{aligned} g(Pre(f_1), \ldots, Pre(f_n), Post(f_1), \ldots, Post(f_n)) = \\ \sum_{i=1}^{m} \sum_{j=1}^{n} [Pre(p_i, f_j) + Post(p_i, f_j)] \end{aligned} \tag{3.9}$$

The solution of the identification problem can be computed by solving the following integer programming problem (IPP), assuming that the initial marking μ_0 *is given:*

$$\begin{cases} \min & g(Pre(f_1), \ldots, Pre(f_n), Post(f_1), \ldots, Post(f_n)) \\ s.t. & G_m(PL(N)^f, PNL(N)^f) \end{cases}$$

In this way, they determine a Petri net that minimizes the sum of the weights of the arcs from a place to a faulty transition or from a faulty transition to a place.

The following example, Example 21, shows how we can solve Problem **RQ1** when applying Theorem 19 and Proposition 20.

Example 21. Consider the Petri net in Figure 3.1 to represent the fault-free system.

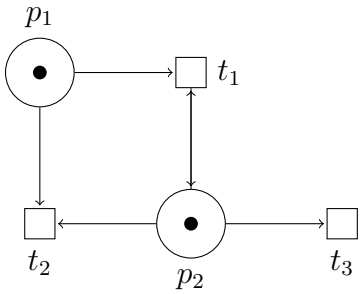

Figure 3.1: The fault-free net system.

The pre and post condition and the initial marking of the Petri net are

$$Pre = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}, Post = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \mu_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Let $L_q(N, \mu_0) = \{\epsilon, t_1, t_2, t_3, t_1t_3\}$ where $q = 2$ be its language. Now let $T_f = \{f_1\}$ and $L^f = \{\epsilon, t_1, t_2, t_3, t_1t_3, f_1, f_1t_3\}$ be the language of the faulty system. Assume that we want to find a Petri net system N^f that minimizes the sum of the arc weights on the fault transitions as in Proposition 20 such that $L_q(N^f, \mu_0) = L^f$. We can solve this problem by using the sets

$$PL(N)^f = \{(\epsilon, f_1), (f_1, t_3)\}$$

and

$$PNL(N)^f = \{(\ f_1, t_1), (f_1, t_2), (f_1, f_1), (t_1, f_1), (t_2, f_1), (t_3, f_1)\}$$

where the pre and post condition and the initial marking of the faulty Petri net are

$$Pre^f = \begin{bmatrix} 1 & 1 & 0 & x_1 \\ 1 & 1 & 1 & x_2 \end{bmatrix}, Post^f = \begin{bmatrix} 0 & 0 & 0 & y_1 \\ 1 & 0 & 0 & y_2 \end{bmatrix}, \mu_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

where $x_1 = \omega(p_1, f_1), x_2 = \omega(p_2, f_1), y_1 = \omega(f_1, p_1), y_2 = \omega(f_1, p_2)$ are the weights of the arcs from a place to the faulty transition and vice-versa.

In order to find the pre and post conditions of the faulty transition we solve the first inequality of Theorem 19. Let us consider each element in the

set $PL(N)^f$, because the first inequality holds for all pairs in $PL(N)^f$. For example, for the first pair (ϵ, f_1) we have that

$$\mu_0 + Post \cdot \pi(\sigma) - Pre \cdot [\pi(\sigma) + \vec{t_j}] \geq \vec{0}$$

$$\begin{bmatrix}1\\1\end{bmatrix} + \begin{bmatrix}0 & 0 & 0 & y_1\\1 & 0 & 0 & y_2\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\end{bmatrix} - \begin{bmatrix}1 & 1 & 0 & x_1\\1 & 1 & 1 & x_2\end{bmatrix} \left[\begin{pmatrix}0\\0\\0\\0\end{pmatrix} + \begin{pmatrix}0\\0\\0\\1\end{pmatrix} \right] \geq \begin{bmatrix}0\\0\end{bmatrix},$$

$$\begin{bmatrix}1\\1\end{bmatrix} - \begin{bmatrix}x_1\\x_2\end{bmatrix} \geq \begin{bmatrix}0\\0\end{bmatrix}.$$

Doing the algebra we get the inequalities

$$1 - x_1 \geq 0$$

$$1 - x_2 \geq 0$$

Now, consider the second pair (f_1, t_3), so we have that

$$\begin{bmatrix}1\\1\end{bmatrix} + \begin{bmatrix}0 & 0 & 0 & y_1\\1 & 0 & 0 & y_2\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\end{bmatrix} - \begin{bmatrix}1 & 1 & 0 & x_1\\1 & 1 & 1 & x_2\end{bmatrix} \left[\begin{pmatrix}0\\0\\0\\1\end{pmatrix} + \begin{pmatrix}0\\0\\1\\0\end{pmatrix} \right] \geq \begin{bmatrix}0\\0\end{bmatrix},$$

$$\begin{bmatrix}1\\1\end{bmatrix} + \begin{bmatrix}y_1\\y_2\end{bmatrix} - \begin{bmatrix}x_1\\1 + x_2\end{bmatrix} \geq \begin{bmatrix}0\\0\end{bmatrix}.$$

Doing the algebra we get the inequalities

$$1 + y_1 - x_1 \geq 0$$

$$1 + y_2 - 1 - x_2 \geq 0$$

Now consider the second constraint of Theorem 19 and each one of the pairs in the set $PNL(N)^f$. So for the first pair (f_1, t_1) we have that

$$-KS_{\sigma,j} + \mu_0 + Post \cdot \pi(\sigma) - Pre \cdot [\pi(\sigma) + \vec{t_j}] \leq -\vec{1}_m$$

$$-1\begin{bmatrix}0\\1\end{bmatrix}+\begin{bmatrix}1\\1\end{bmatrix}+\begin{bmatrix}0&0&0&y_1\\1&0&0&y_2\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\end{bmatrix}-\begin{bmatrix}1&1&0&x_1\\1&1&1&x_2\end{bmatrix}\left[\begin{pmatrix}0\\0\\0\\1\end{pmatrix}+\begin{pmatrix}1\\0\\0\\0\end{pmatrix}\right]\leq\begin{bmatrix}-1\\-1\end{bmatrix},$$

$$\begin{bmatrix}0\\-1\end{bmatrix}+\begin{bmatrix}1\\1\end{bmatrix}+\begin{bmatrix}y_1\\y_2\end{bmatrix}-\begin{bmatrix}1+x_1\\1+x_2\end{bmatrix}\leq\begin{bmatrix}-1\\-1\end{bmatrix}.$$

Now, consider the second pair (f_1, t_2), so we have that

$$-1\begin{bmatrix}0\\1\end{bmatrix}+\begin{bmatrix}1\\1\end{bmatrix}+\begin{bmatrix}0&0&0&y_1\\1&0&0&y_2\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\end{bmatrix}-\begin{bmatrix}1&1&0&x_1\\1&1&1&x_2\end{bmatrix}\left[\begin{pmatrix}0\\0\\0\\1\end{pmatrix}+\begin{pmatrix}0\\1\\0\\0\end{pmatrix}\right]\leq\begin{bmatrix}-1\\-1\end{bmatrix},$$

$$\begin{bmatrix}0\\-1\end{bmatrix}+\begin{bmatrix}1\\1\end{bmatrix}+\begin{bmatrix}y_1\\y_2\end{bmatrix}-\begin{bmatrix}1+x_1\\1+x_2\end{bmatrix}\leq\begin{bmatrix}-1\\-1\end{bmatrix}.$$

For the third pair (f_1, f_1) we have that

$$-1\begin{bmatrix}0\\1\end{bmatrix}+\begin{bmatrix}1\\1\end{bmatrix}+\begin{bmatrix}0&0&0&y_1\\1&0&0&y_2\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\end{bmatrix}-\begin{bmatrix}1&1&0&x_1\\1&1&1&x_2\end{bmatrix}\left[\begin{pmatrix}0\\0\\0\\1\end{pmatrix}+\begin{pmatrix}0\\0\\0\\1\end{pmatrix}\right]\leq\begin{bmatrix}-1\\-1\end{bmatrix},$$

$$\begin{bmatrix}0\\-1\end{bmatrix}+\begin{bmatrix}1\\1\end{bmatrix}+\begin{bmatrix}y_1\\y_2\end{bmatrix}-\begin{bmatrix}2x_1\\2x_2\end{bmatrix}\leq\begin{bmatrix}-1\\-1\end{bmatrix}.$$

Now, consider the forth pair (t_1, f_1), so we have that

$$-1\begin{bmatrix}0\\1\end{bmatrix}+\begin{bmatrix}1\\1\end{bmatrix}+\begin{bmatrix}0&0&0&y_1\\1&0&0&y_2\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\end{bmatrix}-\begin{bmatrix}1&1&0&x_1\\1&1&1&x_2\end{bmatrix}\left[\begin{pmatrix}1\\0\\0\\0\end{pmatrix}+\begin{pmatrix}0\\0\\0\\1\end{pmatrix}\right]\leq\begin{bmatrix}-1\\-1\end{bmatrix},$$

$$\begin{bmatrix}0\\-1\end{bmatrix}+\begin{bmatrix}1\\1\end{bmatrix}+\begin{bmatrix}0\\1\end{bmatrix}-\begin{bmatrix}1+x_1\\1+x_2\end{bmatrix}\leq\begin{bmatrix}-1\\-1\end{bmatrix}.$$

Doing the algebra we get the inequalities

$$y_1 - x_1 \leq -1$$

$$y_2 \geq x_2$$

$$y_1 - 2x_1 \leq -1$$

By solving these inequalities and applying the minimization we have

$$x_1 = 1$$

$$x_2 \leq 1$$

$$y_1 \geq 0$$

$$y_2 \geq 1 \geq x_2$$

Now, consider the third constraint of Theorem 19 and each one of the pairs in the set $PNL(N)^f$. So we have that

$$\vec{1}^T S_{\sigma,j} \leq m - 1$$

If $\mu_\sigma(p_i) < Pre(p_i, t_j)$ then $s_i = 0$ otherwise $s_i = 1$. So for pair (f_1, t_1) we have that

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} < \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The resulting matrix for $S_{\sigma,j}$ is the following

$$S_{f_1,t_1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

So for the inequality above we have that

$$\vec{1}^T \begin{bmatrix} 0 \\ 1 \end{bmatrix} \leq 2 - 1\,,$$
$$1 \leq 1$$

Similarly for the other pairs of the set $PNL(N)^f$. We can clearly see that all the constraints are satisfied.

The resulting matrices corresponding to the faulty system are then

$$Pre^f = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, Post^f = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}, \mu_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

This gives the net shown in Figure 3.2; as it can be noted in this figure, now it is clear that faulty transition is connect to P_1 and P_2. This piece of information will be very useful in the subsequent debugging of the implementation and will guide the process by pinpointing the nature of the introduced fault.

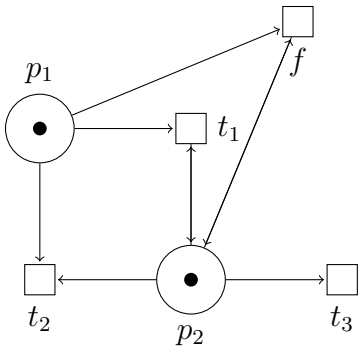

Figure 3.2: The resulting of applying fault identification to the Petri Net of Figure 3.1 and using the traces in Example 21.

□

Chapter 4

Unobservable fault

4.1 Background

In the previous chapters, we assumed that the set of faulty transitions are known and they are observable in the sequences of the faulty net. In this chapter, we relax this assumption and assume the presence of an unobservable fault, which without loss of generality is assume to be represented by a transition f with unknown pre- and post-conditions. Unobservability means that this transition does not show up in the sequences of the faulty net and its effect has to be inferred from enabledness and disabledness of other observable and non-faulty transitions.

In the next chapter, I revisit a similar result due to Cabasino [14] and point out a few issues that we observed in the old result and show how I have fixed these issues in my results.

4.2 Recapitulating problem definition

In this section we recall Problem **(RQ2)** in Section 2.6.2, which is studied in the remainder of this chapter.

Consider a fault-free system $\langle N, \mu_0 \rangle$ with $L(N)$ its language, where T, i.e., the set of transitions of N, is the alphabet of $L(N)$. Consider $\langle N^f, \mu_0 \rangle$ that is the faulty extension of $\langle N, \mu_0 \rangle$. We assume that the faulty transition(s) is not observable, but of course its presence may influence the firing of other transitions in T.

Assuming that we are provided with the specification of $\langle N, \mu \rangle$ and also

the set of firing sequences $L^f_{\leq q}$ of the faulty net (in which the fault itself is abstracted away).

Following Cabasino [14], we make the following assumption.

Assumption 22. The faulty net contains a single fault, i.e., $k = 1$. For simplicity, we denote the faulty transition by f, hence $T^f = T \cup \{f\}$. □

Assumption 23. The transition f is loop-free, i.e., $Pre(f) \cap Post(f) = \emptyset$. □

The latter assumption is not overly restrictive as the effect of loops can always be cancelled out; however, it comes in handy when making calculations in the following fault identification problem.

The problem studied in this chapter is to identify the fault f, by identifying its pre- and post-conditions, $\Downarrow_T (L_{\leq q}(N^f)) = L^f_{\leq q}$.

Let $P = \{p_1, p_2, \ldots, p_n\}$ be the set of places of N and N^f, and choose an ordering $(p_1, p_2, \ldots, p_n)$.

4.3 Identifying an unobservable fault

Theorem 24. *Let $\langle N^f, \mu_0 \rangle$ be a marked Petri net where $N = (P, T, F, \omega)$ and μ_0 is the initial marking and let $\langle N^f, \mu_0 \rangle$ be the faulty extension of N with T^f under Assumptions 22 and 23. Suppose that $\sigma \in T^*$ is such that $\Downarrow_T (\tau) = \sigma$ for some $\tau \in L(N^f)$, where $\Downarrow_T$ is the projection from $(T^f)^*$ onto T^* as defined in Definition 14. Let M be the minimum number of occurrences of f which occur in such sequences τ and let $t_j \in T$. Then the following are equivalent:*

(i) $\sigma t_j = \Downarrow_T (\psi)$ for some $\psi \in L(N^f)$;

(ii) There exists $l \geq M$ such that

$$\mu_0 + C \cdot \pi(\sigma) + l \cdot [Post(f) - Pre(f)] \geq Pre(t_j) \tag{4.1}$$

We want to prove that statement (i) holds if and only if statement (ii) holds. We prove this in terms of two implications.

Proof. We first want to prove that if (i) holds, then (ii) must hold as well.

Suppose that $\sigma t_j = \Downarrow_T (\psi)$ for some $\psi \in L(N^f)$. Starting at the marking μ_0, we can fire a sequence τ in N^f with exactly M occurrences of f such that $\Downarrow_T (\tau) = \sigma$ to reach a marking μ in N^f where

$$\mu = \mu_0 + C \cdot \pi(\sigma) + M \cdot [Post(f) - Pre(f)]. \tag{4.2}$$

We then fire a subsequent sequence ϕ in N^f such that $\Downarrow_T (\phi) = t_j$.

Without loss of generality, we may assume that $\phi = f^r t_j$ for some $r \geq 0$ (ϕ must be of the form $f^r t_j f^s$ for some $r, s \geq 0$, and we may simply delete f^s if $s > 0$).

Now let $l = M + r \geq M$. Since $f^r t_j$ is enabled at μ in N^f, we must have that

$$\mu + r \cdot [Post(f) - Pre(f)] \geq Pre(t_j). \tag{4.3}$$

Combining Equations 4.2 and 4.3 yields that

$$\mu_0 + C \cdot \pi(\sigma) + M \cdot [Post(f) - Pre(f)] + r \cdot [Post(f) - Pre(f)] \geq Pre(t_j), \tag{4.4}$$

which gives (ii) as required (since $l = M + r \geq M$).

Now we want to prove that if (ii) holds, then (i) must hold as well.

Suppose that there exists $l \geq M$ such that

$$\mu_0 + C \cdot \pi(\sigma) + l \cdot [Post(f) - Pre(f)] \geq Pre(t_j) \tag{4.5}$$

Set $r = l - M \geq 0$.

We know that τ is enabled at μ_0 in N^f, such that τ contains M occurrences of f and that $\Downarrow_T (\tau) = \sigma$. Firing τ at μ_0 in N^f reaches the marking

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \mu_0 + [Post(\pi(\sigma)) - Pre(\pi(\sigma))] + M \cdot [Post(f) - Pre(f)],$$

where we are taking some fixed ordering of the places. Let

$$[Post(f) - Pre(f)] = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}.$$

By Equation 4.5 we have that

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + (l - M) \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \geq Pre(t_j),$$

and so

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + r \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \geq Pre(t_j) \geq \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{4.6}$$

If we could fire f r times at $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ then we would, in turn, reach the markings

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \xrightarrow{f} \begin{bmatrix} x_1 + a_1 \\ x_2 + a_2 \\ \vdots \\ x_n + a_n \end{bmatrix} \xrightarrow{f} \begin{bmatrix} x_1 + 2a_1 \\ x_2 + 2a_2 \\ \vdots \\ x_n + 2a_n \end{bmatrix} \xrightarrow{f} \cdots \xrightarrow{f} \begin{bmatrix} x_1 + ra_1 \\ x_2 + ra_2 \\ \vdots \\ x_n + ra_n \end{bmatrix}.$$

We see that we can fire f r time at $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ provided that $x_i + ja_i \geq x_i \geq 0$ for $1 \leq i \leq n$ and $0 \leq j \leq r$.

If $a_i \geq 0$ then it is clear that $x_i + ja_i \geq x_i \geq 0$ for $0 \leq j \leq r$.

If $a_i < 0$ then we have that

$$x_i > x_i + a_i > x_i + 2a_i > \cdots > x_i + ra_i \geq 0$$

by Equation 4.6.

So we can fire f r time at $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ and we can reach the marking

$$\mu = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + r \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}.$$

By Equation 4.6 we see that t_j is enabled at μ in N^f. Letting $\psi = \tau f^r t_j \in L(N^f)$, we have that $\sigma t_j = \Downarrow_T (\psi)$ as required.

□

Theorem 25. *Let $\langle N^f, \mu_0 \rangle$ be a marked Petri net where $N = (P, T, F, \omega)$ and μ_0 is the initial marking and let $\langle N^f, \mu_0 \rangle$ be the faulty extension of N with T^f under Assumptions 22 and 23. Suppose that R is a prefix-closed language with $L(N) \subseteq R \subseteq T^*$. Then the following are equivalent:*

(i) $\Downarrow_T (L(N^f)) = R$

(ii) If $t_j \in T$, $\sigma \in \Downarrow_T (L(N^f))$ and M is the minimum number of occurrences of f in all sequences $\tau \in L(N^f)$ such that $\Downarrow_T (\tau) = \sigma$, then:

$$\sigma t_j \in R \iff \text{there exists } l \geq M \text{such that } \mu_0 + C \cdot \pi(\sigma) + l \cdot [Post(f) - Pre(f)] \geq Pre(t_j)$$

We want to prove that statement (i) holds if and only if statement (ii) holds. We prove this in terms of two implications.

Proof. We first want to prove that if (i) holds, then (ii) must hold as well.

Suppose that (i) holds, i.e. that $\Downarrow_T (L(N^f)) = R$.

By Theorem 24 we have that, if $t_j \in T$, $\sigma \in \Downarrow_T (L(N^f))$ and M is the minimum number of occurrences of f in all sequences $\tau \in L(N^f)$ such that $\Downarrow_T (\tau) = \sigma$, then:

$$\sigma t_j \in \Downarrow_T (L(N^f)) \iff \text{there exists } l \geq M \text{such that } \mu_0 + C \cdot \pi(\sigma) + l \cdot [Post(f) - Pre(f)] \geq Pre(t_j).$$

Since $\Downarrow_T (L(N^f)) = R$, part (ii) follows immediately.

Now we want to prove that if (ii) holds, then (i) must hold as well.

Assume that (ii) holds. Let $L = \Downarrow_T (L(N^f))$; we want to show that $L = R$. We will do this by showing that $L_{\leq}q = R_{\leq}q$ for all $q \in \mathbb{N}$; we proceed by induction on q.

If $q = 0$ then $L_{\leq}q = \{\lambda\} = R_{\leq}q$ (note that $\lambda \in L$ by definition and $\lambda \in R$ as R is non-empty and prefix-closed).

Now assume that $L_{\leq}q = R_{\leq}q$ for some $q \geq 0$. Let $\alpha \in T^*$ with $|\alpha| = q+1$, say $\alpha = \sigma t_j$ with $|\sigma| = q$ and $t_j \in T$.

Since L and R are both prefix-closed, if $\sigma \notin L_{\leq q} = R_{\leq q}$, so that $\sigma \notin L$ and $\sigma \notin R$, then $\sigma t_j \notin L$ and $\sigma t_j \notin R$, so that $\sigma t_j \notin L_{\leq(q+1)}$ and $\sigma t_j \notin R_{\leq(q+1)}$.

So let us assume that $\sigma \in L_{\leq q} = R_{\leq q}$. Let M be the minimum number of occurrences of f in sequences τ in $L(N^f)$ such that $\Downarrow_T (\tau) = \sigma$ (this makes sense as $\sigma \in L = \Downarrow_T (L(N^f))$). Now we have that:

$$\begin{aligned}\alpha = \sigma t_j \in R \iff & \text{ there exists } l \geq M \text{such that} \\ & \mu_0 + C \cdot \pi(\sigma) + l \cdot [Post(f) - Pre(f)] \geq Pre(t_j) \quad \text{by part (ii)} \\ & \iff \alpha = \sigma t_j \in L \qquad \text{by Theorem 24}\end{aligned}$$

Thus $L_{\leq(q+1)} = R_{\leq(q+1)}$ as required.

By induction we have shown that $L = R$, and hence part (i) holds. □

In a similar vein we have:

Theorem 26. *Let $\langle N^f, \mu_0 \rangle$ be a marked Petri net where $N = (P, T, F, \omega)$ and μ_0 is the initial marking and let $\langle N^f, \mu_0 \rangle$ be the faulty extension of N with T^f under Assumptions 22 and 23. Suppose that R is a prefix-closed language with $L_{\leq q}(N) \subseteq R \subseteq T^*_{\leq q}$ for some $q > 0$. Then the following are equivalent:*

(i) $\Downarrow_T (L_{\leq q}(N^f)) = R$

(ii) If $t_j \in T$, $\sigma \in \Downarrow_T (L_{\leq q-1}(N^f))$ and M is the minimum number of occurrences of f in all sequences $\tau \in L(N^f)$ such that $\Downarrow_T (\tau) = \sigma$, then:

$$\begin{aligned}\sigma t_j \in R \iff & \textit{ there exists } l \geq M \textit{such that} \\ & \mu_0 + C \cdot \pi(\sigma) + l \cdot [Post(f) - Pre(f)] \geq Pre(t_j).\end{aligned}$$

We want to prove that statement (i) holds if and only if statement (ii) holds. We prove this in terms of two implications.

Proof. We first want to prove that if (i) holds, then (ii) must hold as well.

This is the same as for the proof of Theorem 25.

Now we want to prove that if (ii) holds, then (i) must hold as well.

This is essentially the same as for the proof of Theorem 25 but our induction stops with the step from $q-1$ to q (so that we don't need the full principle of induction here, we just go $0 \to 1 \to 2 \to \cdots \to q$).

□

Example 27. If N is the Petri net

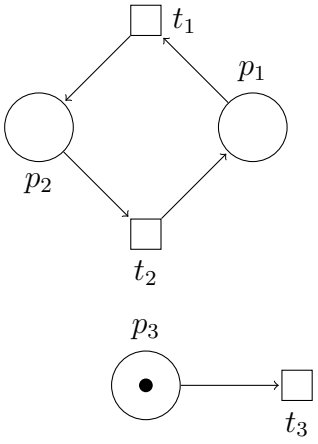

Figure 4.1: The fault-free net system.

with initial marking $\mu_0 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ (as shown) and N^f is the following faulty extension of N:

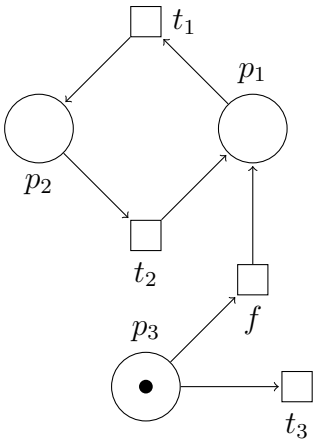

Figure 4.2: The faulty net system.

It is clear that $L(N) = L_{\leq 3}(N) = \{\lambda, t_3\}$.

Let us consider $\Downarrow_T (L_{\leq 3}(N^f))$. We can determine this by constructing part of the reachability tree for N^f, in Figure 4.3:

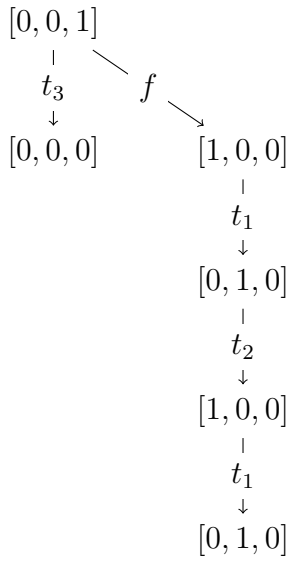

Figure 4.3: Reachability tree

$$t_1 \leftrightarrow \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}; \qquad t_3 \leftrightarrow \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix};$$

$$t_2 \leftrightarrow \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}; \qquad f \leftrightarrow \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}.$$

Note that f can only fire once at the beginning of a sequence of transitions and is then permanently disabled; so all sequences in $\Downarrow_T (L_{\leq 3}(N^f))$ have been accounted for and we have:

$$\Downarrow_T (L_{\leq 3}(N^f)) = \{\lambda, t_3, t_1, t_1t_2, t_1t_2t_1\}$$

.

In the next example we will take the language $\Downarrow_T (L_{\leq 3}(N^f))$ as given here as our starting point and see if we can deduce the structure of N^f.

□

Example 28. Consider the net N in Example 27 with the same initial marking. We saw in Example 27 that $L(N) = \{\lambda, t_3\}$. We now specify a language $L = \{\lambda, t_3, t_1, t_1t_2, t_1t_2t_1\}$ and see if we can determine which faulty extensions of N, where we add a single loop-free fault f, give rise to this language L if we consider sequences of length at most 3, i.e. which faulty extensions N^f of N satisfy $\Downarrow_T (L_{\leq 3}(N^f)) = L$.

Let

$$Post(f) - Pre(f) = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

Note that, as f is loop-free, we must have:

$$\begin{aligned} &x_i = 0 \Rightarrow Pre(p_i, f) = Post(p_i, f) = 0; \\ &x_i > 0 \Rightarrow Pre(p_i, f) = 0, Post(p_i, f) = x_i > 0; \\ &x_i < 0 \Rightarrow Pre(p_i, f) = -x_i > 0, Post(p_i, f) = 0. \end{aligned} \tag{4.7}$$

We will use the equivalence of the conditions specified in Theorem 24. Consider the pair (ϵ, t_1). Here $\sigma = \lambda$ and $M = 0$.

$$\begin{aligned} t_1 \in \Downarrow_T (L(N^f)) &\Longrightarrow \exists\, l \geq 0 : \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + l \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \geq \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ &\Longrightarrow \exists\, l \geq 1 : \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + l \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \geq \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ &\qquad (l = 0 \text{ clearly does not satisfy the inequality}) \\ &\Longrightarrow \exists\, l \geq 1 :\ lx_1 \geq 1,\ lx_2 \geq 0,\ 1 + lx_3 \geq 0 \\ &\Longrightarrow \exists\, l \geq 1 :\ x_1 \geq 1,\ x_2 \geq 0,\ lx_3 \geq -1 \\ &\Longrightarrow x_1 \geq 1,\ x_2 \geq 0,\ x_3 \geq -1 \ (x_3 < -1 \Longrightarrow lx_3 < -1 \text{ for all } l \geq 1) \end{aligned} \tag{4.8}$$

Now consider the pair (t_3, t_1). Here $\sigma = t_3$ and $M = 0$.

$$\begin{aligned} t_3 t_1 \notin \Downarrow_T (L(N^f)) &\Longrightarrow \forall\, l \geq 0 : \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} + l \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \not\geq \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ &\Longrightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \not\geq \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ (putting } l = 1) \\ &\Longrightarrow x_1 < 1,\ \textit{or}\ x_2 < 0,\ \textit{or}\ x_3 < 0 \end{aligned} \tag{4.9}$$

Combining conditions (4.8) and (4.9) we see that we must have that $x_3 < 0$ in condition (4.9), as $x_1 < 1$ and $x_2 < 0$ are impossible by (4.8), and so $-1 \leq x_3 < 0$, giving that $x_3 = -1$. So we now have that

$$Post(f) - Pre(f) = \begin{bmatrix} x_1 \\ x_2 \\ -1 \end{bmatrix} \text{ with } x_1 \geq 1 \text{ and } x_2 \geq 0.$$

Consider the pair (ϵ, t_2). Here $\sigma = \lambda$ and $M = 0$.

$$\begin{aligned} t_2 \notin \Downarrow_T (L(N^f)) &\Longrightarrow \forall\, l \geq 0 : \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + l \begin{bmatrix} x_1 \\ x_2 \\ -1 \end{bmatrix} \not\geq \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \\ &\Longrightarrow \begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix} \not\geq \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ (putting } l = 1) \\ &\Longrightarrow x_2 < 1 \text{ (as we know that } x_1 \geq 1) \\ &\Longrightarrow x_2 = 0 \text{ (as we now know that } 0 \leq x_2 < 1). \end{aligned}$$

So we now have that

$$Post(f) - Pre(f) = \begin{bmatrix} x_1 \\ 0 \\ -1 \end{bmatrix} \text{ with } x_1 \geq 1.$$

Now $t_1 \notin L(N)$ but $t_1 \in \Downarrow_T (L(N^f))$. What is the minimum number M of times we need to fire f at μ_0 to enable t_1 in N^f? Clearly $M > 0$ and then

$$\begin{aligned}\mu_0 + Post(f) - Pre(f) &= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} x_1 \\ 0 \\ -1 \end{bmatrix} \\ &= \begin{bmatrix} x_1 \\ 0 \\ 0 \end{bmatrix} \\ &\geq \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ &= Pre(t_1).\end{aligned}$$

So for $t_1 t_1 \notin \Downarrow_T (L(N^f))$, we have that $\sigma = t_1$ and $M = 1$, and then

$$\begin{aligned}t_1 t_1 \notin \Downarrow_T (L(N^f)) &\Longrightarrow \forall\, l \geq 1 : \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} + l \begin{bmatrix} x_1 \\ 0 \\ -1 \end{bmatrix} \not\geq \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ &\Longrightarrow \begin{bmatrix} x_1 - 1 \\ 1 \\ 0 \end{bmatrix} \not\geq \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ (putting } l = 1) \\ &\Longrightarrow x_1 - 1 < 1 \\ &\Longrightarrow x_1 < 2.\end{aligned}$$

So $1 \leq x_1 \leq 2$, and so we must have that $x_1 = 1$. We now have that

$$Post(f) - Pre(f) = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix},$$

and so, given our observations in (4.7) above, we must have that

$$Pre(f) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad Post(f) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

This means that the only possible faulty extension N^f of N with $\Downarrow_T (L_{\leq 3}(N^f)) = L$ is the Petri net

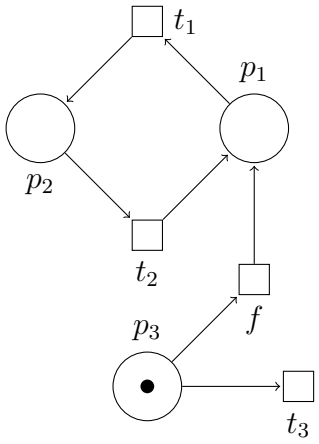

Figure 4.4: The faulty net system.

in Example 27 above. □

Chapter 5

Revisiting unobservable fault identification

5.1 Background

Our approach to fault identification is largely based on an earlier approach by Cabasino [14]. However, upon a careful examination of her approach we found a number of shortcomings that led us to its redesign in the previous chapter. In this chapter, we review the original proof by Cabasino and point out these shortcomings. In Cabasino's work there exists an introduction to linear- programming, that was used for the solution of Problem 16 as it was described in her thesis [14]. Later in this chapter, we show an alternative translation to linear programming and we show how this is equivalent to Cabasino's work.

5.2 Connection to Cabasino's approach

Let $\langle N^f, \mu_0 \rangle$ be a marked Petri net where $N = (P, T, F, \omega)$ and μ_0 is the initial marking and let $\langle N^f, \mu_0 \rangle$ be the faulty extension of N with T^f under Assumptions 22 and 23. Suppose that $\sigma \in T^*$ is such that $\Downarrow_T (\tau) = \sigma$ for some $\tau \in L(N^f)$, where $\Downarrow_T$ is the projection from $(T^f)^*$ onto T^* as defined in Definition 14. Let M be the minimum number of occurrences of f which occur in such sequences τ and let $t_j \in T$. As in Theorem 24 we have that the following are equivalent:

$$\exists \psi \in L(N^f) : \sigma t_j = \Downarrow_T (\psi);$$
$$\exists l \geq M : \mu_0 + C \cdot \pi(\sigma) + l \cdot [Post(f) - Pre(f)] \geq Pre(t_j).$$

Given this, we see that the following are equivalent:

$$\nexists \psi \in L(N^f) : \sigma t_j = \Downarrow_T (\psi);$$
$$\forall l \geq M : \mu_0 + C \cdot \pi(\sigma) + l \cdot [Post(f) - Pre(f)] \ngeq Pre(t_j).$$

Let us take a fixed ordering $(p_1, p_2, \ldots, p_n)$ of the elements of P. For $1 \leq i \leq n$ and $l \geq 0$, we define

$$R(p_i, l) = \mu_0(p_i) + C(p_i) \cdot \pi(\sigma) + l \cdot [Post(p_i, f) - Pre(p_i, f)] - Pre(p_i, t_j).$$

Then condition

$$\forall l \geq M : \mu_0 + C \cdot \pi(\sigma) + l \cdot [Post(f) - Pre(f)] \ngeq Pre(t_j) \tag{5.1}$$

is equivalent to:

$$\forall l \geq M, \exists i : R(p_i, l) < 0.$$

For $1 \leq i \leq n$ and $l \geq 0$ we now define

$$u_{i,l} = \begin{cases} 1 & \text{if } R(p_i, l) \geq 0 \\ 0 & \text{if } R(p_i, l) < 0. \end{cases}$$

We see that Condition 5.1 is now equivalent to:

$$\forall l \geq M, \exists i : u_{i,l} = 0,$$

which, in turn, is equivalent to

$$\forall l \geq M : \begin{bmatrix} u_{1,l} \\ u_{2,l} \\ \vdots \\ u_{n,l} \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix},$$

which is equivalent to

$$\forall l \geq M, u_{1,l} + u_{2,l} + \cdots + u_{n,l} \leq n - 1,$$

and then to

$$\forall l \geq M, (1, 1, \ldots, 1) \begin{bmatrix} u_{1,l} \\ u_{2,l} \\ \vdots \\ u_{n,l} \end{bmatrix} \leq n - 1.$$

Let K_l denote

$$max\{R(p_1, l), R(p_2, l), \ldots, R(p_n, l)\} + 1.$$

Then for $1 \leq i \leq n$, we have

$$\begin{aligned} R(p_i, l) < 0 &\Longrightarrow u_{i,l} = 0 \\ &\Longrightarrow R(p_i, l) - K_l u_{i,l} = R(p_i, l) < 0, \\ R(p_i, l) \geq 0 &\Longrightarrow u_{i,l} = 1 \\ &\Longrightarrow R(p_i, l) < K_l = K_l u_{i,l} \\ &\Longrightarrow R(p_i, l) - K_l u_{i,l} < 0. \end{aligned}$$

In the set $\{0, 1\}^n$, we let $\vec{0}$ denote $\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ and $\vec{1}$ denote $\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$. We have shown that Condition 5.1 implies that

$$\forall l \geq M : \begin{bmatrix} R(p_1, l) \\ R(p_2, l) \\ \vdots \\ R(p_n, l) \end{bmatrix} - K_l \begin{bmatrix} u_{1,l} \\ u_{2,l} \\ \vdots \\ u_{n,l} \end{bmatrix} < \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

in other words

$$\forall l \geq M: \; -K_l u_l + \mu_0 + [Post(\pi(\sigma)) - Pre(\pi(\sigma))] \\ + l[Post(f) - Pre(f)] - Pre(t_j) < \vec{0} \quad (5.2)$$

where $\vec{u_l} \in \{0,1\}^n$ and $\vec{1^T} \vec{u_l} \leq n-1$.

Conversely suppose that Condition 5.2 holds. Fix l such that $l \geq M$. Since $\vec{1^T} \vec{u_l} \leq n-1$ we may choose i with $1 \leq i \leq n$ and $u_{i,l} = 0$. For that i we have that $R(p_i, l) = 0$ (by definition of $u_{i,l}$), i.e.

$$\mu_0(p_i) + C(p_i) \cdot \pi(\sigma) + l[Post(p_i, f) - Pre(p_i, f)] - Pre(p_i, t_j) < 0$$

by definition of $R(p_i, l)$. So

$$\mu_0 + C \cdot \pi(\sigma) + l \cdot [Post(f) - Pre(f)] \ngeq Pre(t_j).$$

Since this is true for any fixed $l \geq M$ we have that

$$l \geq M: \; \mu_0 + C \cdot \pi(\sigma) + l[Post(f) - Pre(f)] \ngeq Pre(t_j),$$

which is Condition 5.1. So conditions 5.1 and 5.2 are equivalent. We sum this up in the following result:

Theorem 29. *Let $\langle N^f, \mu_0 \rangle$ be a marked Petri net where $N = (P, T, F, \omega)$ and μ_0 is the initial marking and let $\langle N^f, \mu_0 \rangle$ be the faulty extension of N with T^f under Assumptions 22 and 23. Suppose that $\sigma \in T^*$ is such that $\Downarrow_T (\tau) = \sigma$ for some $\tau \in L(N^f)$, where $\Downarrow_T$ is the projection from $(T^f)^*$ onto T^* as defined in Definition 14. Let M be the minimum number of occurrences of f which occur in such sequences τ and let $t_j \in T$. Then*

$$\sigma t_j \in \Downarrow_T (L(N^f)) \Longrightarrow \exists l \geq M \\ \mu_0 + C \cdot \pi(\sigma) + l[Post(f) - Pre(f)] \geq Pre(t_j);$$

$$\sigma t_j \notin \Downarrow_T (L(N^f)) \Longrightarrow \forall l \geq M, \exists \vec{u_l} \in \{0,1\}^n \text{ and } K_l > 0 : \vec{1^T} \vec{u_l} \leq n-1 \text{ and} \\ - K_l \vec{u_l} + \mu_0 + C \cdot \pi(\sigma) \\ + l[Post(f) - Pre(f)] - Pre(t_j) < \vec{0}.$$

There is one last part to Cabasino's characterization in [14] which captures the fact that f is loop-free which is as follows:

$$\exists \text{ constant } R > 0 \text{ and vectors } \vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}, \vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \in \{0,1\}^n \text{ such that}$$

$$\begin{aligned} u_i + v_1 &= 1 (1 \leq i \leq n), \\ Pre(p_i, f) - Ru_i &\leq 0 (1 \leq i \leq n), \\ Post(p_i, f) - Rv_i &\leq 0 (1 \leq i \leq n). \end{aligned} \tag{5.3}$$

The fact that $\vec{u}$ and $\vec{v}$ must lie in $\{0,1\}^n$ is implicit in 14 but is not explicitly stated there.

Below is the proof that Condition 5.3 implies that f is loop-free.

Proof. If we fix i with $1 \leq i \leq n$, then

$$\begin{aligned} u_i + v_i = 1, u_i \in \{0,1\}, v_i \in \{0,1\} &\Longrightarrow u_i = 0 \textit{ or } v_i = 0 \\ &\Longrightarrow Ru_i = 0 \textit{ or } Rv_i = 0 \\ &\Longrightarrow Pre(p_i, f) = Pre(p_i, f) - Ru_i \leq 0 \text{ or} \\ &\quad\; Post(p_i, f) = Post(p_i, f) - Rv_i \leq 0 \\ &\Longrightarrow Pre(p_i, f) = 0 \textit{ or } Post(p_i, f) = 0. \end{aligned}$$

Since this it true for all i with $1 \leq i \leq n$ we see that f is loop-free. □

Now we want to prove that f being loop-free implies Condition 5.3

Proof. Assume that f is loop-free.

For $1 \leq i \leq n$ let

$$\begin{aligned} u_i &= \begin{cases} 1 & \text{if } Pre(p_i, f) > 0 \\ 0 & \text{if } Pre(p_i, f) = 0 \end{cases}, \\ v_i &= 1 - u_i. \end{aligned}$$

Then:

$$\begin{aligned} Pre(p_i, f) > 0 &\Longrightarrow u_i = 1, v_i = 0 \text{ (by definition)} \\ Post(p_i, f) > 0 &\Longrightarrow Pre(p_i, f) = 0 \text{ (since f is loop-free)} \\ &\Longrightarrow u_i = 0, v_i = 1 \text{ (by definition)} \end{aligned}$$

Let

$$R = \max\{Pre(p_1, f), Pre(p_2, f), \ldots, Pre(p_n, f), \\ Post(p_1, f), Post(p_2, f), \ldots, Post(p_n, f)\}.$$

Then:

$$\begin{aligned} Pre(p_i, f) > 0 &\Longrightarrow u_i = 1 \\ &\Longrightarrow Pre(p_i, f) - Ru_i \leq 0 \text{ (by definition of R);} \\ Pre(p_i, f) = 0 &\Longrightarrow u_i = 0 \\ &\Longrightarrow Pre(p_i, f) - Ru_i = 0. \end{aligned}$$

So $Pre(p_i, f) - Ru_i \leq 0$ for $1 \leq i \leq n$.

$$\begin{aligned} Post(p_i, f) > 0 &\Longrightarrow v_i = 1 \\ &\Longrightarrow Post(p_i, f) - Rv_i \leq 0 \text{ (by definition of R);} \\ Post(p_i, f) = 0 &\Longrightarrow Post(p_i, f) - Rv_i = -Rv_i \leq 0. \end{aligned}$$

So $Post(p_i, f) - Rv_i \leq 0$ for $1 \leq i \leq n$ as required. □

5.3 Issues with Cabasino's approach

In this section we address the issues of Cabasino's approach and we suggest some improvements.

Let us separate out the two directions of Theorem 24:

$$\begin{aligned} \exists \psi \in L(N^f) : \sigma t_j = \Downarrow_T (\psi) \Longrightarrow &\exists l \geq M : \mu_0 + C \cdot \pi(\sigma) \\ &+ l \cdot [Post(f) - Pre(f)] \geq Pre(t_j). \end{aligned} \tag{5.4}$$

$$\begin{aligned} &\exists l \geq M : \mu_0 + C \cdot \pi(\sigma) \\ &\quad + l \cdot [Post(f) - Pre(f)] \geq Pre(t_j) \Longrightarrow \exists \psi \in L(N^f) : \sigma t_j = \Downarrow_T (\psi). \end{aligned} \tag{5.5}$$

In her version of 5.4 in [14] Cabasino omits the condition that $l \geq M$, simply stating that:

$$\begin{aligned} \exists \psi \in L(N^f) : \sigma t_j = \Downarrow_T (\psi) \Longrightarrow &\exists l : \mu_0 + C \cdot \pi(\sigma) \\ &+ l \cdot [Post(f) - Pre(f)] \geq Pre(t_j). \end{aligned}$$

This is, of course, still true. The issues is with her approach relating to 5.5.

We have seen that 5.5 is equivalent to:

$$\begin{aligned}\nexists\psi \in L(N^f) : \sigma t_j = \Downarrow_T (\psi) \Longrightarrow \forall l \geq M, \exists \vec{u_l} \in \{0,1\}^n \text{ and } K_l \geq 0 : \\ 1^{\vec{T}}\vec{u_l} \leq n-1 \text{ and} \\ -K_l\vec{u_l} + \mu_0 + [Post(\pi(\sigma)) - Pre(\pi(\sigma))] \\ + l[Post(f) - Pre(f)] - Pre(t_j) < \vec{0}.\end{aligned}$$

Cabasino appears to have instead:

$$\begin{aligned}\nexists\psi \in L(N^f) : \sigma t_j = \Downarrow_T (\psi) \Longrightarrow \exists \vec{u} \in \{0,1\}^n \text{ and } K > 0 : 1^{\vec{T}}\vec{u} \leq n-1 \text{ and} \\ \forall l \geq 0, -K\vec{u} + \mu_0 + [Post(\pi(\sigma)) - Pre(\pi(\sigma))] \\ + l[Post(f) - Pre(f)] - Pre(t_j) < \vec{0}.\end{aligned}$$

Theere are two potential issues here:

1. Having K independent of l;
2. Having $\vec{u}$ independent of l.

We shall treat these issues in turn and show that there are problems with each formulation.

Issue 1: Having K independent of l. As we have said, in Cabasino's approach in [14], the constant K in the condition

$$-K\vec{u} + \mu_0 + C \cdot \pi(\sigma) + l[Post(f) - Pre(f)] - Pre(t_j) < \vec{0}$$

appears to be independent of l. In our approach we had:

$$R(p_i, l) := \mu_0(p_i) + C(p_i) \cdot \pi(\sigma) + l \cdot [Post(p_i, f) - Pre(p_i, f)] - Pre(p_i, t_j);$$

$$K_l := max\{R(p_1, l), R(p_2, l), \ldots, R(p_n, l)\} + 1.$$

With this approach it is clear that, if $Post(p_i, f) - Pre(p_i, f) > 0$ for some i (i.e. if $Post(p_i, f) > 0$ as f is loop-free), then $R(p_i, l) \to \infty$ ad $l \to \infty$, and so $K_l \to \infty$ as $l \to \infty$. With our approach we see that K_l is not bounded and we cannot replace it by a single constant K.

Of course, it might seem that a differnet approach could remove this issue of K not being independent of l. To see that there is a genuine problem here,

let us consider a specific instance of this problem. Consider the Petri net N and the faulty extension N^f of N shown below where the initial marking μ_0 is $\begin{bmatrix}1\\1\end{bmatrix}$:

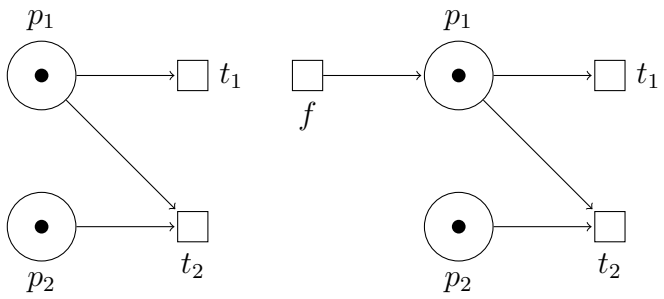

Figure 5.1: The fault-free and the faulty net systems.

It is clear that $\mathcal{L}(N) = \{\lambda, t_1, t_2\}$. With regards to N^f, we can enable t_1 at any stage (firing f if necessary), but t_2 can be fired (at most) once and, if fired, is then permanently disabled. So

$$\Downarrow_T (L(N^f)) = \{t_1\}^* \cup \{t_1\}^*\{t_2\}\{t_1\}^*.$$

Let $\sigma = t_2 \in \mathcal{L}(N)$; so $M = 0$ in this case. Now $\sigma t_2 = t_2 t_2 \notin \Downarrow_T (L(N^f))$. According to [14] we would have:

$$\exists \vec{u} \in \{0,1\}^n \text{ and } K > 0 \text{ such that } \vec{1^T}\vec{u} \leq n-1 \text{ and}$$

$$\forall l \geq 0, -K\vec{u} + \mu_0 + [Post(\pi(\sigma)) - Pre(\pi(\sigma))] + l[Post(f) - Pre(f)] - Pre(t_j) < \vec{0},$$

which becomes

$$\exists \vec{u} = \begin{bmatrix}u_1\\u_2\end{bmatrix} \in \{0,1\}^n \text{ and } K > 0 \text{ such that } \vec{1^T}\vec{u} \leq n-1 \text{ and } \forall l \geq 0,$$

$$-K\begin{bmatrix}u_1\\u_2\end{bmatrix} + \begin{bmatrix}1\\1\end{bmatrix} + \left[\begin{pmatrix}0\\0\end{pmatrix} - \begin{pmatrix}1\\1\end{pmatrix}\right] + l\left[\begin{pmatrix}1\\0\end{pmatrix} - \begin{pmatrix}0\\0\end{pmatrix}\right] - \begin{bmatrix}1\\1\end{bmatrix} < \begin{bmatrix}0\\0\end{bmatrix},$$

and then

$$\exists \vec{u} = \begin{bmatrix}u_1\\u_2\end{bmatrix} \in \{0,1\}^n \text{ and } K > 0 \text{ such that } \vec{1^T}\vec{u} \leq n-1 \text{ and } \forall l \geq 0,$$

$$\begin{bmatrix}l - Ku_1 - 1\\ -Ku_2 - 1\end{bmatrix} < \begin{bmatrix}0\\0\end{bmatrix}.$$

This cannot hold for all l if K is fixed.

Remark 30. In our discussion of Issue 1 out example demonstrating the problem, the faulty extension N^f satisfies the condition

$$Pre(p, f) = 0 \text{ for all } p \in P. \tag{5.6}$$

We will discuss here how to deal with possibilities satisfying 5.6 and thereby reduce the problem to one in which Issue 2 is no longer a problem.

Let us assume that we have our usual set-up:

$$N = (P, T, F, \omega) \text{ with } \mu_0; \;\; N^f = (P, T^f, F^f, \omega^f) \text{ with } \mu_0;$$

$$T^f = T \cup \{f\}; \;\; f \notin T \text{ and } f \text{ loop-free },$$

with N^f satisfying 5.6. The problem here arises form the fact that, because of 5.6, f can fire any number of times at any stage. We will consider an alternative approach to this particular situation.

Let $P_1 = \{p \in P : Post(P) > 0\}$. In our faulty extension N^f any pre-conditions of a transition which lie in P_1 are essentially irrelevant as we can fire f sufficiently many times (without any restrictions) to satisfy all of these (i.e., for any place p, to put sufficiently many tokens n_p in place p such that $n_p \geq Pre(p, t)$ for all $t \in T$).

So what we have is effectively a new Petri net with the places in P_1 (and any arcs involving them) removed. We can then remove f to get a new net N' with $\mathcal{L}(N') = \Downarrow_T (L(N^f))$. We obviously need to change μ_0 to reflect the fact that some places have been removed (the markings in other places in μ_0 remain the same).

Looking for possible faulty extensions N^f of N satisfying 5.6 is equivalent, therefore, to considering all Petri nets N' obtained from N where we delete a subset P_1 of P (and all arcs involving the places in P_1). Since there are only $2^{|P|}$ possibilities for P_1 and we are comparing finit languages in each case (namely $\mathcal{L}(N)_q$ and $\mathcal{L}(N')_q$ fro some q), we have an algorithm for finding all faulty extensions of this type.

Let us illustrate this with the Petri net N and the faulty extension N^f we used to illustrate Issue 1:

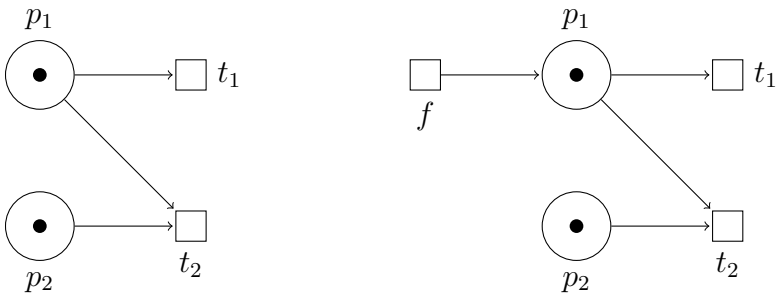

Figure 5.2: The fault-free and the faulty net systems.

□

In this case $P_1 = \{p_1\}$. Removing p_1 (and all the arcs involved with it) from N yields the Petri net N' shown below:

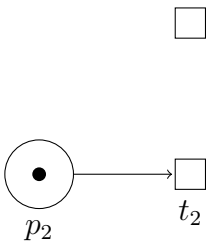

Figure 5.3: The new Petri net N'.

Note that $\mathcal{L}(N') = \{t_1\}^* \cup \{t_1\}^*\{t_2\}\{t_1\}^*$ (which is the same as that of $\Downarrow_T (L(N^f)$ inthe discussion of Issue 2 above).

Of course, this onlu finds possibilities for the faulty extenstion N^f where 5.6 holds. What about other possibilities, i.e. faulty extensions which satisfy $Pre(p, f) > 0$ for some $p \in P$?

So suppose we have a place p with $Pre(p, f) > 0$ (and then, by our assumption that f is loop-free, we must have that $Post(p, f) = 0$). Then, in any sequence $\tau \in (T^f)^*$ with $\Downarrow_T (\tau) = \sigma$, the maximum number of times f can fire in τ is bounded above by b where

$$b = \frac{\mu_0(p) + Post(p, \sigma)}{Pre(p, f)}.$$

The point is that we can accumulate at most $\mu_0(p) + Post(p, \sigma)$ tokens in place p given the initial tokens there in μ_0 anf those placed there by firing

the transitions in σ (we may not be able to achieve that many tokens in p, but this value b is certainly an upper bound), and firing f uses $Pre(p, f)$ of these tokens each time. This means that, in Condition 5.5, we must have that $l \leq b$ (in additional to $l \geq M$). Recall that we defined

$$\begin{aligned} R(p_i, l) &= \mu_0(p_i) + C(p_i) \cdot \pi(\sigma) + l \cdot [Post(p_i, f) - Pre(p_i, f)] - Pre(p_i, t_j); \\ K_l &= max\{R(p_1, l), R(p_2, l), \ldots, R(p_n, l)\} + 1. \end{aligned}$$

In Cabasino's version we have

$$\begin{aligned} \nexists \psi \in L(N^f) : \sigma t_j = \Downarrow_T (\psi) \Longrightarrow \exists \vec{u} \in \{0,1\}^n \text{ and } K > 0 : \vec{1^T} \vec{u} \leq n-1 \text{ and} \\ \forall l \geq 0, -K\vec{u} + \mu_0 + C \cdot \pi(\sigma) \\ + l[Post(f) - Pre(f)] - Pre(t_j) < \vec{0}. \end{aligned} \tag{5.7}$$

where K is assumed to be independent of l. If we have the extra condition that $l \leq b$ in our formulation, then we may take K in her condition to be

$$\max\{K_M, K_{M+1}, \ldots, K_b\}.$$

The point is that, in 5.7, one can replace a specific value of K by a larger one and 5.7 remains valid.

Taking this approach (finding possibilities fro faulty extensions that satisfy 5.6 first and then considering appropriate bounds for K) allows us to deal with Issue 1 in Cabasino's approach. In addition, Cabasino states (in Remark 10.5 of [14]) that "In practice ... it is sufficient to pick K very large". The approach outlined here allows us to choose a specific value for K in 5.7 which is independent of l. □

Issue 2: Having $\vec{u}$ independent of l.

In Cabasino's approach, the vector $\vec{u}$ is the claim

$$\begin{aligned} \nexists \psi \in L(N^f) : \sigma t_j = \Downarrow_T (\psi) \Longrightarrow \exists \vec{u} \in \{0,1\}^n \text{ and } K > 0 : \vec{1^T} \vec{u} \leq n-1 \text{ and} \\ \forall l \geq 0, -K\vec{u} + \mu_0 + [Post(\pi(\sigma)) - Pre(\pi(\sigma))] \\ + l[Post(f) - Pre(f)] - Pre(t_j) < \vec{0}. \end{aligned}$$

appears to be independent of l.

To see the problem here, consider a Petri net N and faulty extension N^f of N shown below where the initial marking μ_0 is $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$:

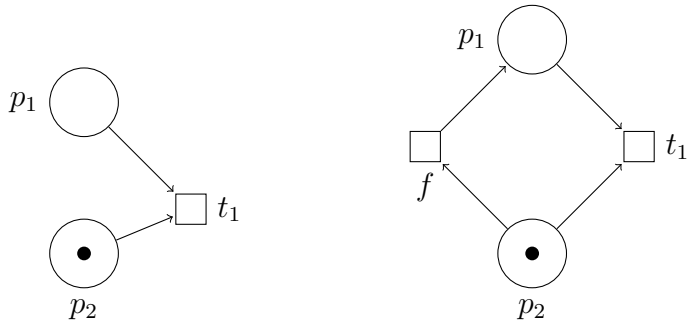

Figure 5.4: The fault-free and the faulty net systems.

It is clear that $\mathcal{L}(N) = \Downarrow_T (L(N^f)) = \{\lambda\}$ here.

Take $\sigma = \lambda$. As $\sigma t_1 \notin \Downarrow_T (L(N^f))$, we should have K and $\vec{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ such that, for all $l \geq 0$,

$$-K \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right] + l \left[\begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right] - \begin{bmatrix} 1 \\ 1 \end{bmatrix} < \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

i.e.

$$\begin{bmatrix} -Ku_1 + l - 1 \\ -Ku_2 - l \end{bmatrix} < \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

If $l = 0$ then we want $-Ku_1 - 1 < 0$ and $-Ku_2 < 0$ and so we must have that

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

If $l = 1$ then we want $-Ku_1 < 0$ and $-Ku_2 - 1 < 0$ and so we must have that

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

So $\vec{u}$ is not independent of l.

In this chapter, we have reviewed the original proof by Cabasino and we have pointed out some shortcomings we found in her approach. In Cabasino's work there exists an introduction to linear- programming, that was used for the solution of Problem 16 as it was described in her thesis [14]. In this chapter, we have shown an alternative translation to linear programming and how this is equivalent to Cabasino's work.

Chapter 6

Implementation

In this chapter we discuss the implementation of our tool, that translates the fault identification problem into a set of linear inequations and subsequently, into a satisfiability modulo theories (SMT) problem. We then use Microsoft Z3 SMT Solver to solve the fault identification problem. We have applied our tool on a number of examples and plan to turn the prototype into a stable tool and perform an empirical evaluation of its performance in the future.

6.1 SAT and SMT solvers: an overview

The past few years have seen an enormous progress in the performance of SAT solvers, enabling formal verification of large and complex systems [5]. A SAT solver is an algorithmic toolkit to solve the NP-Complete problem of Boolean satisfiability problem (SAT), using various powerful heuristics. The SAT problem is to determines if there exists an interpretation of Boolean propositions that satisfies a given Boolean formula. In other words, given a formula, the solver checks whether its variables can be replaced by Boolean values, True or False, such that the formula evaluates to True. We call a formula *satisfiable* if such an interpretation exists, and *unsatisfiable* otherwise [32]. SAT was the first problem that was proven to be NP-complete. Despite their theoretical worst case complexity, SAT solvers have been successfully applied to different problems, such as artificial intelligence problems, circuit design problems, and automatic theorem proving.

These days, SAT almost always refers to CNF-SAT, a boolean satisfaction problem for formulas in conjunctive normal form (CNF). This means that

the entire formula is a conjunction (AND) of clauses, with each clause being a disjunction (OR) of literals [32]. Very few problems are expressed as a logical formula in the CNF format. Due to this fact, after formulating a problem as a SAT, we often need to convert it into CNF and then determine its satisfiability. There are different approaches of converting a SAT problem into CNF. However, one has to be careful as some formulas could blow up exponentially large in the process of CNF conversion.

Microsoft Z3, is a high performance solver that extends satisfiability solving to support other theories such as linear integer arithmetic. It offers various APIs including one in Python, which has been used to implement the theories presented in this thesis. It has in the past been used in many other applications such as verification, testing, constraint solving, security analysis, solving geometrical problems, and the analysis of hybrid and dynamical systems.

6.2 General structure of the prototype

In this section I present the structure of the implementation.

For our implementation, we use automated theorem proving, based on a SAT solver extended with theories, namely Microsoft Z3. Z3 provides different APIs and solvers; we use its API in Python and its basic Solver.

The idea is to create a simple tool that codes Theorem 24 into an SMT problem and solves it using Z3 to characterise a faulty transition that can explain the faulty language L^f. The traces are coded in our simple syntax: each trace is coded as a matrix representing the appearance of transitions in that trace. This facilitates the subsequent step of coding the specific problem into the in-equations specified by Theorem 24. Once this step is perfomed the result is passed to Z3 and it is checked whether this system of inequations is satisfiable. If yes, then Z3 automatically finds a satisfying interpretation of the pre- and post-conditions of the fault; otherwise it shows that the requirements have not been satisfied and that there is no solution for such a net. We have implemented this process in an initial prototype, described below; we intend to turn this into a full-fledged tool for identifying multiple faults in the future.

6.3 Coding and parsing Petri Nets and traces

To start with, we need to code the structure of the Petri net and the traces into a formal syntax. These files include all the information needed to construct a Petri net, i.e., the initial marking, pre- and post conditions of transitions, the firing sequences of trances along with the ones that are found not to belong to the faulty language.

We use the example in Figure 6.1 to illustrate our syntax.

```
places
2
p1
p2
transitions
3
t1
t2
t3
mu0
1
1
pre
1 1 0
1 1 1
post
0 0 0
1 0 0
in_Lf(5)
t1
[0,0,0]
t2
[0,0,0]
t3
[0,0,0]
t1,t3
[1,0,0]
t3,t3
[0,0,1]
not_in_Lf(4)
t1,t1
[1,0,0]
t1,t2
[1,0,0]
t3,t1
[0,0,1]
```

```
t3,t2
[0,0,1]
```

Figure 6.1: The structure of the index file.

As it can be seen above, first the basic structure of the specification Petri net are specified; to facilitate parsing, we start with the number of places and transitions and then specify their names. Subsequently, one can specify the initial marking for places and pre- and post-condition matrices for transitions. Finally, one specifies the bounded traces that belong, and do not belong to the faulty language, respectively. To guide the testing of our tool, we have created a number of examples in this language, which we have subsequently used in testing various steps of our implementation. We plan to develop this set of examples into a benchmark inspired by realistic specifications and faults.

After having defined our input syntax, we read the specification of the Petri net and the traces of the faulty net and generate a Python script that codes the fault identification problem. The code snippet in Figure 6.2 represents the parsing step in Python.

```
#!/bin/python

import ast
import sys
import re

places = []
transitions = []
mu0 = {}
pre = {}
post = {}
in_Lf = {}
not_in_Lf = {}

try:
fname = sys.argv[1]
f = open(fname,'r')

# Read number of places and its IDs
f.readline().strip() # read line with 'places'
n_places = int(f.readline().strip())
for i in range(n_places):
line = f.readline().strip()
```

```
places.append(line)
f.readline().strip() # jump blank line

# Read number of transitions and its IDs
f.readline().strip() # read line with 'transitions'
n_trs = int(f.readline().strip())
for i in range(n_trs):
line = f.readline().strip()
transitions.append(line)
f.readline().strip() # jump blank line

# Read the initial markings
f.readline().strip() # read line with 'mu0'
for i in range(n_places):
line = f.readline().strip()
mu0[places[i]] = line
f.readline().strip() # jump blank line

# Read the matrix of pre-conditions
f.readline().strip() # read line with 'pre'
for _pn in range(n_places):
line = f.readline().strip()
line = line.split(' ')
pre[places[_pn]] = {}
# print(line)
for _tn in range(n_trs):
pre[places[_pn]][transitions[_tn]] = line[_tn]
f.readline().strip() # jump blank line

# Read the matrix of pre-conditions
f.readline().strip() # read line with 'post'
for _pn in range(n_places):
line = f.readline().strip()
line = line.split(' ')
post[places[_pn]] = {}
# print(line)
for _tn in range(n_trs):
post[places[_pn]][transitions[_tn]] = line[_tn]
f.readline().strip() # jump blank line

# Read the list of sequences in_Lf
f.readline().strip() # read line with 'in_Lf'
sz_Lf = int(f.readline().strip())
for _sn in range(sz_Lf):
_seq = f.readline().strip().split(',')
```

```
69 f.readline()
70 in_Lf[_sn] = _seq
71 f.readline().strip() # jump blank line
72
73 # Read the list of sequences not_in_Lf
74 f.readline().strip() # read line with 'not_in_Lf'
75 sz_not_in_Lf = int(f.readline().strip())
76 for _sn in range(sz_not_in_Lf):
77 _seq = f.readline().strip().split(',')
78 f.readline()
79 not_in_Lf[_sn] = _seq
80 f.readline().strip() # jump blank line
81 f.close()
```

Figure 6.2: Code snippet in Python for Parsing the Petri net information and the Traces of the Faulty System.

6.4 Coding the fault identification problem in Z3

In the next step, we initialize the constraints needed to solve the identification problem. To do this, we generate a Python script that can make a call to the Z3 Sovler API and re-construct the problem structure in this Python script.

The code snippet presented in Figure 6.3 is the first step of this code generation, where the basic structure of the Petri net is reconstructed in the generated Python script. In this code snippet, we first initialize the constraints and then we read the index file line by line. We first read the numbers of places and then each place itself, and then we read the number of transitions and each transition itself. Afterwards, we read the initial marking, and the pre and post conditions of each transition. Then, the only things left to read is the faulty language of the Petri net along with the language that it is not. We have split up the language into a list of sequences, so that the program deals with one sequence at a time. We read the sequences that are in L^f and then the sequences that are not in L^f. Note that we specify the number of sequences each time, in order to calculate how many sequences exist in each set and then generate the inequations accordingly. Also, note that we separate each transition in the sequence using ",", in order to show that the prexif of the sequence is σ and that the remaining transition is the t_j.

Then we initialize the constraints needed to solve the identification problem. To do this, we generate a Python script that can make a call to the Z3 Sovler API and re-construct the problem structure in this Python script. Code snippet presented in Figure 6.3 is the first step of this code generation, where the basic structure of the Petri net is reconstructed in the generated Python script.

After reading the index file, we start generating a Python script by first importing Z3 solver, in order to find a solution to our problem. We then translate all the information gathered from the index file to a Python script and store them in the generated file. Figure 6.3 represents the process of storing the initial marking and the matrices of the pre- and post-conditions for each transition.

```
# 'Starting to print the script'
print('#!/bin/python')
print('')
print('from z3 import *')
print('')
print('# we have that')
print('s = Solver()')
print('## mu0_px is the initial marking for place px; ')
print('%s = %s' % (\
', '.join(['mu_'+places[_pn] for _pn in range(n_places)]), \
', '.join([mu0[places[_pn]] for _pn in range(n_places)])) \
)

print('')
print('## pi_tj is the pre-condition from place pi to
    transition tj')
for _pn in range(n_places):
print('%s = %s' % (\
', '.join([places[_pn]+'_'+transitions[_tn] for _tn in range(
    n_trs)]), \
', '.join([pre[places[_pn]][transitions[_tn]] for _tn in
    range(n_trs)])) \
)
print('')
print('## tj_pi is the post-condition from transition tj to
    place pi')
for _pn in range(n_places):
print('%s = %s' % (\
', '.join([transitions[_tn]+'_'+places[_pn] for _tn in range(
    n_trs)]), \
', '.join([post[places[_pn]][transitions[_tn]] for _tn in
```

```
    range(n_trs)]))
)
```

Figure 6.3: Coding the Initial marking, and Pre- and Post Conditions in the Generated File for Z3.

Subsequently, we define the unknowns of the fault identification problem, namely the pre- and post- conditions of the faulty transition and the number of firings of the faulty transition in the traces given in the problem definition. The code snippet presented in Figure 6.4, represents this step.

```
print('')
print('## find the values for the faulty transitions ')
for _pn in range(n_places):
print('f_%s, %s_f = Ints(\'f_%s %s_f\')' % (places[_pn],
    places[_pn],places[_pn],places[_pn]))
print('')
print('# where they should be ')
print('s.add( %s )' % ', '.join('f_%s >= 0' % places[_pn] for
     _pn in range(n_places)))
print('s.add( %s )' % ', '.join('%s_f >= 0' % places[_pn] for
     _pn in range(n_places)))
print('')
print('## l \\in Naturals ; ')
for _sn in range(sz_Lf): print('l'+str(_sn)+' = Int(\'l'+str(
    _sn)+'\')')
for _sn in range(sz_not_in_Lf): print('l'+str(_sn+sz_Lf)+' =
    Int(\'l'+str(_sn+sz_Lf)+'\')')
print('')
```

Figure 6.4: Coding the Unknowns of the Fault Identification Problem in Z3.

The next step is to code the constraints of Theorem 24. To do this, we need to add the assumptions required, i.e., there is a single fault, and the fault is loop-free. Once these assumptions are coded, our tool generates different inequalities for different sequences. Note that the sequences can either belong to the enabled or the disabled constraints and based on that we have different values for the variables used and different inequalities for each sequence. It is worth mentioning here that for the enabled constraints we use the **AND** operator as we want all of the inequalities to be satisfied, where in the disabled constraints we use the **OR** operator as at least one place must disable the faulty transition, whilst at least one inequality need to be satisfied. The code snippet in Figure 6.5 represents the creation of the inequations for the enabled constraints, e.g. the sequences that are in L^f.

```
print('########################################')
print('## \\in L^f (Equation 4.1)')
print('########################################')
for _sn in range(sz_Lf):
# set counters for each symbol to zero
_counter = {transitions[_tn]:0 for _tn in range(n_trs)}
_counter['f'] = 0 # set counters for f to zero
# count for all symbols, except the last
for _idx in range(len(in_Lf[_sn])-1):
_counter[in_Lf[_sn][_idx]]+=1
# get the last symbol
last_transition = in_Lf[_sn][-1]
# print(_counter)
print('# Sequence %d: %s' % (_sn,','.join(in_Lf[_sn])))
print('%s = %s' % (\
', '.join(['s'+str(_sn)+'_'+transitions[_tn] for _tn in range
   (n_trs)]), \
', '.join([ str(_counter[transitions[_tn]]) for _tn in range(
   n_trs)]))
)
# print('s.add( l%d >= %d )' % (_sn,_counter['f']))
# print('')
print('s.add(')
print('   Exists([l%d],'%_sn)
print('      And( Implies(l%d >= %d, ' % (_sn,_counter['f']))
print('      And(l%d >= %d, ' % (_sn,_counter['f']))
for _pn in range(n_places):
line = '         mu_'+places[_pn]+' + '
line += ' + '.join(['('+transitions[_tn]+'_'+places[_pn]+'-'+
   places[_pn]+'_'+transitions[_tn]+')*s'+str(_sn)+'_'+
   transitions[_tn] for _tn in range(n_trs)])
line += ' + '+'l'+str(_sn)+' * ('+'f_'+places[_pn]+' - '+
   places[_pn]+'_f'+')'
line += ' >= '+places[_pn]+'_'+last_transition+','
print(line)
print('      )')
print('      ))')
print('   )')
print(')')
print('')
```

Figure 6.5: Coding the Conditions on the Enabled Transitions in Z3.

We first separate the enabled from the disabled constraints and then we translate the enabled constraints from Theorem 24 into linear algebraic inequation in Python. According to the Theorem 24

If $\sigma t_j \in L^f$, then for some $l \in \mathbb{N}$, where $l \geq M$

$$\mu_0(p) + C(p) \cdot \pi(\sigma) + l \cdot [Post(p, f) - Pre(p, f)] \geq Pre(p, t_j)$$

for all $p \in P$.

In order to translate it in Python, we first state that there exists some variable l, as the statement should hold for some l. Due to the fact that these inequations have to be true for all places we use the **AND** operator and without loss of generality we translate the above equation in terms of a Python script. Note that each generated inequation corresponds to a place.

Similarly, for the disabled constraints, i.e., for the sequences not in L^f, we have that

If $\sigma t_j \notin L^f$, then for all $l \in \mathbb{N}$, where $l \geq M$

$$\mu_0(p) + C(p) \cdot \pi(\sigma) + l \cdot [Post(p, f) - Pre(p, f)] < Pre(p, t_j)$$

for some $p \in P$.

In order to translate this into Python code, we use the same procedure as before with the difference that we use the **OR** operator, due to the fact that these inequations have to be true for at least one place. Moreover, note that the inequations here must hold for all l and for that reason we use the keyword **ForAll**. The figure below shows what was mentioned above.

```
print('')
print('########################################')
print('## \\not \\in L^f (Equation 4.2)')
print('########################################')
for _sn in range(sz_not_in_Lf):
# set counters for each symbol to zero
_counter = {transitions[_tn]:0 for _tn in range(n_trs)}
_counter['f'] = 0 # set counters for f to zero
# count for all symbols, except the last
for _idx in range(len(not_in_Lf[_sn])-1):
_counter[not_in_Lf[_sn][_idx]]+=1
# get the last symbol
last_transition = not_in_Lf[_sn][-1]
# print(_counter)
print('# Sequence %d: %s' % (_sn+sz_Lf,','.join(not_in_Lf[_sn
    ])))
print('%s = %s' % (\
', '.join(['s'+str(_sn+sz_Lf)+'_'+transitions[_tn] for _tn in
    range(n_trs)]), \
```

```
', '.join([ str(_counter[transitions[_tn]]) for _tn in range(
    n_trs)]))
)
# print('s.add( l%d >= %d )' % (_sn+sz_Lf,_counter['f']))
# print('')
print('s.add(')
print('   ForAll([l%d],' % (_sn+sz_Lf))
print('      And( Implies(l%d >= %d, ' % (_sn+sz_Lf,_counter[
    'f']))
print('      Or(')
for _pn in range(n_places):
line = '          And(l%d >= %d, ' % (_sn+sz_Lf,_counter['f'])
line += ' mu_'+places[_pn]+' + '
line += ' + '.join(['('+transitions[_tn]+'_'+places[_pn]+'-'+
    places[_pn]+'_'+transitions[_tn]+')*s'+str(_sn+sz_Lf)+'_'+
    transitions[_tn] for _tn in range(n_trs)])
line += ' + '+'l'+str(_sn+sz_Lf)+' * ('+'f_'+places[_pn]+' -
    '+places[_pn]+'_f'+')'
line += ' < '+places[_pn]+'_'+last_transition+'),'
print(line)
print('      )')
print('      ))')
print('   )')
print(')')
print('')
print('')
```

Figure 6.6: Implementation of the disabled constraints in the base file.

Last but not least, we call two methods in order to check and view a representation of the calculated result. The first method is called **check()** and is used in order to solve the asserted constraints. The second method is called **model()** and is used in order to provide a model for the last check. The following figure represents the two methods called.

```
# print('print(s)')
print('print(s.check())')
print('print(s.model())')
```

Figure 6.7: Commands to solve and provide the faulty net system in the base file.

When the Python script that uses Z3 solver is generated, we can then run it to see if the result is SAT or unSAT. A model exists if and only if the constraints are satisfied, otherwise no model is available.

In Figure 6.8, we present an example of the Python script generated by our tool. As explained before, we first import the Z3 solver and then we use the command **Solver()** which creates a general purpose solver, in order to find a solution to our problem. We then set up the constraints of Theorem 24. Namely, the initial marking, the firing sequences, the pre and post conditions of each transition, and the unknowns, which are the number of firings of the faulty transition along with its pre and post conditions. Moreover, we initialize some variables l_i, which represent the occurrences of the faulty transition in each sequence. A representation can be found below.

```
#!/bin/python

from z3 import *

# we have that
s = Solver()
## mu0_px is the initial marking for place px;
mu_p1, mu_p2 = 1, 1

## pi_tj is the pre-condition from place pi to transition tj
p1_t1, p1_t2, p1_t3 = 1, 1, 0
p2_t1, p2_t2, p2_t3 = 1, 1, 1

## tj_pi is the post-condition from transition tj to place pi
t1_p1, t2_p1, t3_p1 = 0, 0, 0
t1_p2, t2_p2, t3_p2 = 1, 0, 0

## find the values for the faulty transitions
f_p1, p1_f = Ints('f_p1 p1_f')
f_p2, p2_f = Ints('f_p2 p2_f')

# where they should be
s.add( f_p1 >= 0, f_p2 >= 0 )
s.add( p1_f >= 0, p2_f >= 0 )

## l \in Naturals ;
l0 = Int('l0')
l1 = Int('l1')
l2 = Int('l2')
l3 = Int('l3')
l4 = Int('l4')
l5 = Int('l5')
l6 = Int('l6')
l7 = Int('l7')
```

```
35 l8 = Int('l8')
```

Figure 6.8: An Example of the Generated Python Script Representing a Fault Identification Problem.

In addition, we assert the constraints of the theorem in the solver by using the **add()** method. The first constraint asserted is based on the assumption where there is a single fault, and the second assertion is with regards to the self-loopness of the faulty transition, as shown in the figure below.

```
s.add(And(
  Or( And((f_p1 >= 0, p1_f == 0)),
    And((f_p1 == 0, p1_f >= 0)),
    And((f_p1 == 0, p1_f == 0))),
  Or( And((f_p2 >= 0, p2_f == 0)),
    And((f_p2 == 0, p2_f >= 0)),
    And((f_p2 == 0, p2_f == 0)))
  )
)
```

Figure 6.9: Python representation of Assumption 22 and Assumption 23.

The assertion procedure is also applied for both enabled and disabled constraints. The inequations below are generated from the base file and the information gathered from the index file.

```
########################################
## \in L^f (Equation 4.1)
########################################
# Sequence 0: t1
s0_t1, s0_t2, s0_t3 = 0, 0, 0
s.add(Exists([l0],
And( Implies(l0 >= 0,
And(l0 >= 0,
  mu_p1 + (t1_p1-p1_t1)*s0_t1 + (t2_p1-p1_t2)*s0_t2
  + (t3_p1-p1_t3)*s0_t3 + l0 * (f_p1 - p1_f) >= p1_t1,
  mu_p2 + (t1_p2-p2_t1)*s0_t1 + (t2_p2-p2_t2)*s0_t2
  + (t3_p2-p2_t3)*s0_t3 + l0 * (f_p2 - p2_f) >= p2_t1,
)))))

# Sequence 1: t2
s1_t1, s1_t2, s1_t3 = 0, 0, 0
s.add(Exists([l1],
And( Implies(l1 >= 0,
And(l1 >= 0,
```

```
  mu_p1 + (t1_p1-p1_t1)*s1_t1 + (t2_p1-p1_t2)*s1_t2
  + (t3_p1-p1_t3)*s1_t3 + l1 * (f_p1 - p1_f) >= p1_t2,
  mu_p2 + (t1_p2-p2_t1)*s1_t1 + (t2_p2-p2_t2)*s1_t2
  + (t3_p2-p2_t3)*s1_t3 + l1 * (f_p2 - p2_f) >= p2_t2,
)))))

# Sequence 2: t3
s2_t1, s2_t2, s2_t3 = 0, 0, 0
s.add(Exists([l2],
And( Implies(l2 >= 0,
And(l2 >= 0,
  mu_p1 + (t1_p1-p1_t1)*s2_t1 + (t2_p1-p1_t2)*s2_t2
  + (t3_p1-p1_t3)*s2_t3 + l2 * (f_p1 - p1_f) >= p1_t3,
  mu_p2 + (t1_p2-p2_t1)*s2_t1 + (t2_p2-p2_t2)*s2_t2
  + (t3_p2-p2_t3)*s2_t3 + l2 * (f_p2 - p2_f) >= p2_t3,
)))))

# Sequence 3: t1,t3
s3_t1, s3_t2, s3_t3 = 1, 0, 0
s.add(Exists([l3],
And( Implies(l3 >= 0,
And(l3 >= 0,
  mu_p1 + (t1_p1-p1_t1)*s3_t1 + (t2_p1-p1_t2)*s3_t2
  + (t3_p1-p1_t3)*s3_t3 + l3 * (f_p1 - p1_f) >= p1_t3,
  mu_p2 + (t1_p2-p2_t1)*s3_t1 + (t2_p2-p2_t2)*s3_t2
  + (t3_p2-p2_t3)*s3_t3 + l3 * (f_p2 - p2_f) >= p2_t3,
)))))

# Sequence 4: t3,t3
s4_t1, s4_t2, s4_t3 = 0, 0, 1
s.add(Exists([l4],
And( Implies(l4 >= 0,
And(l4 >= 0,
  mu_p1 + (t1_p1-p1_t1)*s4_t1 + (t2_p1-p1_t2)*s4_t2
  + (t3_p1-p1_t3)*s4_t3 + l4 * (f_p1 - p1_f) >= p1_t3,
  mu_p2 + (t1_p2-p2_t1)*s4_t1 + (t2_p2-p2_t2)*s4_t2
  + (t3_p2-p2_t3)*s4_t3 + l4 * (f_p2 - p2_f) >= p2_t3,
)))))


########################################
## \not \in L^f (Equation 4.2)
########################################
# Sequence 5: t1,t1
s5_t1, s5_t2, s5_t3 = 1, 0, 0
```

```
s.add(ForAll([l5],
And( Implies(l5 >= 0,
Or(
And(l5 >= 0,
  mu_p1 + (t1_p1-p1_t1)*s5_t1 + (t2_p1-p1_t2)*s5_t2
  + (t3_p1-p1_t3)*s5_t3 + l5 * (f_p1 - p1_f) < p1_t1),
And(l5 >= 0,
  mu_p2 + (t1_p2-p2_t1)*s5_t1 + (t2_p2-p2_t2)*s5_t2
  + (t3_p2-p2_t3)*s5_t3 + l5 * (f_p2 - p2_f) < p2_t1),
)))))

# Sequence 6: t1,t2
s6_t1, s6_t2, s6_t3 = 1, 0, 0
s.add(ForAll([l6],
And( Implies(l6 >= 0,
Or(
And(l6 >= 0,
   mu_p1 + (t1_p1-p1_t1)*s6_t1 + (t2_p1-p1_t2)*s6_t2
   + (t3_p1-p1_t3)*s6_t3 + l6 * (f_p1 - p1_f) < p1_t2),
And(l6 >= 0,
  mu_p2 + (t1_p2-p2_t1)*s6_t1 + (t2_p2-p2_t2)*s6_t2
  + (t3_p2-p2_t3)*s6_t3 + l6 * (f_p2 - p2_f) < p2_t2),
)))))

# Sequence 7: t3,t1
s7_t1, s7_t2, s7_t3 = 0, 0, 1
s.add(ForAll([l7],
And( Implies(l7 >= 0,
Or(
And(l7 >= 0,
   mu_p1 + (t1_p1-p1_t1)*s7_t1 + (t2_p1-p1_t2)*s7_t2
   + (t3_p1-p1_t3)*s7_t3 + l7 * (f_p1 - p1_f) < p1_t1),
And(l7 >= 0,
   mu_p2 + (t1_p2-p2_t1)*s7_t1 + (t2_p2-p2_t2)*s7_t2
   + (t3_p2-p2_t3)*s7_t3 + l7 * (f_p2 - p2_f) < p2_t1),
)))))

# Sequence 8: t3,t2
s8_t1, s8_t2, s8_t3 = 0, 0, 1
s.add(ForAll([l8],
And( Implies(l8 >= 0,
Or(
And(l8 >= 0,
   mu_p1 + (t1_p1-p1_t1)*s8_t1 + (t2_p1-p1_t2)*s8_t2
   + (t3_p1-p1_t3)*s8_t3 + l8 * (f_p1 - p1_f) < p1_t2),
```

```
And(l8 >= 0,
  mu_p2 + (t1_p2-p2_t1)*s8_t1 + (t2_p2-p2_t2)*s8_t2
  + (t3_p2-p2_t3)*s8_t3 + l8 * (f_p2 - p2_f) < p2_t2),
)))))
```

Figure 6.10: An example of the generated file showing the enabled and the disabled constraints.

The **check()** method is then used in order to solve the asserted constraints. If a solution is found, the command check returns **sat**, which means that it satisfies the requirements of the specification. The result is **unsat**, if the requirements of the specification have not been satisfied and so no solution exists. Note that, a solver may fail to solve a system of constraints and **unknown** is returned. Finally, the **model()** method is called in order to provide a model for the last check. If the last check is sat then a model is returned, otherwise an exception is thrown and no model is available. See the representation below.

```
print(s.check())
print(s.model())
```

Figure 6.11: An example of the commands generating the solution in the generated file.

The tool has been tested with various examples and in each case the output is SAT and a model is produced. Note that we are not using any minimization algorithm here, but there is always a model that satisfies the specification's constraints.

6.5 Testing the prototype

We have tested our tool on a number of examples. In this section, we present two examples of the application of our tool, starting from the graphical representation of the net systems along with their non-faulty and the faulty language. Then we show the generated Python files representing these examples and finally, present the outcome of the solution generated. We have consciously chosen these examples to present different levels of complexity.

Example 31. Let N be the specification Petri net depicted in Figure 6.12.

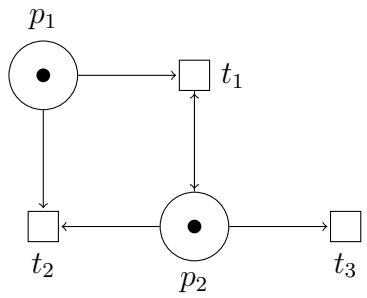

Figure 6.12: The fault-free net system.

The initial marking of N is hence $\mu_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ (as shown).

The Pre and $Post$ conditions of N are

$$Pre = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}, Post = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Let $L_{\leq q}(N) = \{\lambda, t_1, t_2, t_3, t_1t_3\}$ and $L^f_{\leq q} = \{\lambda, t_1, t_2, t_3, t_1t_3, t_2t_3, t_3t_1, t_3t_2, t_3t_3\}$, where q=3. Also let the sequences not in L^f be $\{t_1t_1, t_1t_2, t_2t_1, t_2t_2\}$. The input files to our tool according to our syntax are presented in Figure 6.13.

```
places
2
p1
p2
transitions
3
t1
t2
t3
mu0
1
1
pre
1 1 0
1 1 1
post
0 0 0
1 0 0
in_Lf(8)
```

```
t1
[0,0,0]
t2
[0,0,0]
t3
[0,0,0]
t1,t3
[1,0,0]
t2,t3
[0,1,0]
t3,t3
[0,0,1]
t3,t1
[0,0,1]
t3,t2
[0,0,1]
not_in_Lf(4)
t1,t1
[1,0,0]
t1,t2
[1,0,0]
t2,t1
[0,1,0]
t2,t2
[0,1,0]
```

Figure 6.13: The index file based on the fault-free net.

We can solve this problem by using the sets

$$PL(N)_{\leq q} = \{(\epsilon, t_1), (\epsilon, t_2), (\epsilon, t_3), (t_1, t_3)\}$$

$$PL(N)^f_{\leq q} = \{(t_2, t_3), (t_3, t_1), (t_3, t_2), (t_3, t_3)\}$$

and

$$PNL(N)^f_{\leq q} = \{(t_1, t_1), (t_1, t_2), (t_2, t_1), (t_2, t_2)\}$$

The generated file by our tool is given in Figure 6.14.

```
#!/bin/python

from z3 import *

# we have that
s = Solver()
## mu0_px is the initial marking for place px;
```

```
mu_p1, mu_p2 = 1, 1

## pi_tj is the pre-condition from place pi to transition tj
p1_t1, p1_t2, p1_t3 = 1, 1, 0
p2_t1, p2_t2, p2_t3 = 1, 1, 1

## tj_pi is the post-condition from transition tj to place pi
t1_p1, t2_p1, t3_p1 = 0, 0, 0
t1_p2, t2_p2, t3_p2 = 1, 0, 0

## find the values for the faulty transitions
f_p1, p1_f = Ints('f_p1 p1_f')
f_p2, p2_f = Ints('f_p2 p2_f')

# where they should be
s.add( f_p1 >= 0, f_p2 >= 0 )
s.add( p1_f >= 0, p2_f >= 0 )

s.add(And(
    Or( And((f_p1 >= 0, p1_f == 0)),
        And((f_p1 == 0, p1_f >= 0)),
        And((f_p1 == 0, p1_f == 0))),
    Or( And((f_p2 >= 0, p2_f == 0)),
        And((f_p2 == 0, p2_f >= 0)),
        And((f_p2 == 0, p2_f == 0)))
     )
)

## l \in Naturals ;
l0 = Int('l0')
l1 = Int('l1')
l2 = Int('l2')
l3 = Int('l3')
l4 = Int('l4')
l5 = Int('l5')
l6 = Int('l6')
l7 = Int('l7')
l8 = Int('l8')
l9 = Int('l9')
l10 = Int('l10')
l11 = Int('l11')

########################################
## \in L^f (Equation 4.1)
########################################
```

```
# Sequence 0: t1
s0_t1, s0_t2, s0_t3 = 0, 0, 0
s.add(
   Exists([l0],
      And( Implies(l0 >= 0,
      And(l0 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s0_t1 + (t2_p1-p1_t2)*s0_t2
         +(t3_p1-p1_t3)*s0_t3 + l0 * (f_p1 - p1_f) >= p1_t1,
         mu_p2 + (t1_p2-p2_t1)*s0_t1 + (t2_p2-p2_t2)*s0_t2
         +(t3_p2-p2_t3)*s0_t3 + l0 * (f_p2 - p2_f) >= p2_t1,
)))))

# Sequence 1: t2
s1_t1, s1_t2, s1_t3 = 0, 0, 0
s.add(
   Exists([l1],
      And( Implies(l1 >= 0,
      And(l1 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s1_t1 + (t2_p1-p1_t2)*s1_t2
         +(t3_p1-p1_t3)*s1_t3 + l1 * (f_p1 - p1_f) >= p1_t2,
         mu_p2 + (t1_p2-p2_t1)*s1_t1 + (t2_p2-p2_t2)*s1_t2
         +(t3_p2-p2_t3)*s1_t3 + l1 * (f_p2 - p2_f) >= p2_t2,
)))))

# Sequence 2: t3
s2_t1, s2_t2, s2_t3 = 0, 0, 0
s.add(
   Exists([l2],
      And( Implies(l2 >= 0,
      And(l2 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s2_t1 + (t2_p1-p1_t2)*s2_t2
         +(t3_p1-p1_t3)*s2_t3 + l2 * (f_p1 - p1_f) >= p1_t3,
         mu_p2 + (t1_p2-p2_t1)*s2_t1 + (t2_p2-p2_t2)*s2_t2
         +(t3_p2-p2_t3)*s2_t3 + l2 * (f_p2 - p2_f) >= p2_t3,
)))))

# Sequence 3: t1,t3
s3_t1, s3_t2, s3_t3 = 1, 0, 0
s.add(
   Exists([l3],
      And( Implies(l3 >= 0,
      And(l3 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s3_t1 + (t2_p1-p1_t2)*s3_t2
         +(t3_p1-p1_t3)*s3_t3 + l3 * (f_p1 - p1_f) >= p1_t3,
         mu_p2 + (t1_p2-p2_t1)*s3_t1 + (t2_p2-p2_t2)*s3_t2
```

```
         + (t3_p2-p2_t3)*s3_t3 + l3 * (f_p2 - p2_f) >= p2_t3,
)))))

# Sequence 4: t2,t3
s4_t1, s4_t2, s4_t3 = 0, 1, 0
s.add(
   Exists([l4],
      And( Implies(l4 >= 0,
      And(l4 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s4_t1 + (t2_p1-p1_t2)*s4_t2
         + (t3_p1-p1_t3)*s4_t3 + l4 * (f_p1 - p1_f) >= p1_t3,
         mu_p2 + (t1_p2-p2_t1)*s4_t1 + (t2_p2-p2_t2)*s4_t2
         +(t3_p2-p2_t3)*s4_t3 + l4 * (f_p2 - p2_f) >= p2_t3,
)))))

# Sequence 5: t3,t3
s5_t1, s5_t2, s5_t3 = 0, 0, 1
s.add(
   Exists([l5],
      And( Implies(l5 >= 0,
      And(l5 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s5_t1 + (t2_p1-p1_t2)*s5_t2
         +(t3_p1-p1_t3)*s5_t3 + l5 * (f_p1 - p1_f) >= p1_t3,
         mu_p2 + (t1_p2-p2_t1)*s5_t1 + (t2_p2-p2_t2)*s5_t2
         +(t3_p2-p2_t3)*s5_t3 + l5 * (f_p2 - p2_f) >= p2_t3,
)))))

# Sequence 6: t3,t1
s6_t1, s6_t2, s6_t3 = 0, 0, 1
s.add(
   Exists([l6],
      And( Implies(l6 >= 0,
      And(l6 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s6_t1 + (t2_p1-p1_t2)*s6_t2
         +(t3_p1-p1_t3)*s6_t3 + l6 * (f_p1 - p1_f) >= p1_t1,
         mu_p2 + (t1_p2-p2_t1)*s6_t1 + (t2_p2-p2_t2)*s6_t2
         + (t3_p2-p2_t3)*s6_t3 + l6 * (f_p2 - p2_f) >= p2_t1,
)))))

# Sequence 7: t3,t2
s7_t1, s7_t2, s7_t3 = 0, 0, 1
s.add(
   Exists([l7],
      And( Implies(l7 >= 0,
      And(l7 >= 0,
```

```
          mu_p1 + (t1_p1-p1_t1)*s7_t1 + (t2_p1-p1_t2)*s7_t2
          +(t3_p1-p1_t3)*s7_t3 + l7 * (f_p1 - p1_f) >= p1_t2,
          mu_p2 + (t1_p2-p2_t1)*s7_t1 + (t2_p2-p2_t2)*s7_t2
          +(t3_p2-p2_t3)*s7_t3 + l7 * (f_p2 - p2_f) >= p2_t2,
)))))

########################################
## \not \in L^f (Equation 4.2)
########################################
# Sequence 8: t1,t1
s8_t1, s8_t2, s8_t3 = 1, 0, 0
s.add(
   ForAll([l8],
      And( Implies(l8 >= 0,
      Or(
          And(l8 >= 0,  mu_p1 + (t1_p1-p1_t1)*s8_t1
          +(t2_p1-p1_t2)*s8_t2 + (t3_p1-p1_t3)*s8_t3
          +l8 * (f_p1 - p1_f) < p1_t1),
          And(l8 >= 0,  mu_p2 + (t1_p2-p2_t1)*s8_t1
          +(t2_p2-p2_t2)*s8_t2 + (t3_p2-p2_t3)*s8_t3
          + l8 * (f_p2 - p2_f) < p2_t1),
)))))

# Sequence 9: t1,t2
s9_t1, s9_t2, s9_t3 = 1, 0, 0
s.add(
   ForAll([l9],
      And( Implies(l9 >= 0,
      Or(
          And(l9 >= 0,  mu_p1 + (t1_p1-p1_t1)*s9_t1
          + (t2_p1-p1_t2)*s9_t2 + (t3_p1-p1_t3)*s9_t3
          + l9 * (f_p1 - p1_f) < p1_t2),
          And(l9 >= 0,  mu_p2 + (t1_p2-p2_t1)*s9_t1
          + (t2_p2-p2_t2)*s9_t2 + (t3_p2-p2_t3)*s9_t3
          +  l9 * (f_p2 - p2_f) < p2_t2),
)))))

# Sequence 10: t2,t1
s10_t1, s10_t2, s10_t3 = 0, 1, 0
s.add(
   ForAll([l10],
      And( Implies(l10 >= 0,
     Or(
          And(l10 >= 0,  mu_p1 + (t1_p1-p1_t1)*s10_t1
          + (t2_p1-p1_t2)*s10_t2 + (t3_p1-p1_t3)*s10_t3
```

```
          + l10 * (f_p1 - p1_f) < p1_t1),
          And(l10 >= 0,  mu_p2 + (t1_p2-p2_t1)*s10_t1
          + (t2_p2-p2_t2)*s10_t2 + (t3_p2-p2_t3)*s10_t3
          + l10 * (f_p2 - p2_f) < p2_t1),
)))))

# Sequence 11: t2,t2
s11_t1, s11_t2, s11_t3 = 0, 1, 0
s.add(
   ForAll([l11],
      And( Implies(l11 >= 0,
      Or(
          And(l11 >= 0,  mu_p1 + (t1_p1-p1_t1)*s11_t1
          + (t2_p1-p1_t2)*s11_t2 + (t3_p1-p1_t3)*s11_t3
          + l11 * (f_p1 - p1_f) < p1_t2),
          And(l11 >= 0,  mu_p2 + (t1_p2-p2_t1)*s11_t1
          + (t2_p2-p2_t2)*s11_t2 + (t3_p2-p2_t3)*s11_t3
          + l11 * (f_p2 - p2_f) < p2_t2),
)))))

print(s.check())
print(s.model())
```

Figure 6.14: The generated file based on the fault-free net, its language and the faulty language.

After running the Python script, the solution generated is

```
sat
[p2_f = 0, f_p2 = 1, p1_f = 0, f_p1 = 0]
```

Figure 6.15: The generated solution.

And so the graphical representation of the faulty Petri net is

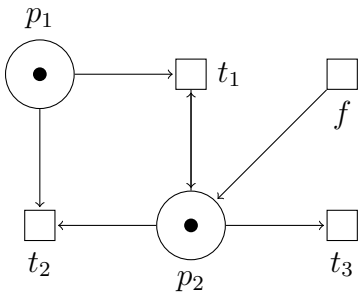

Figure 6.16: The Faulty Net, Automatically Identified by Our Tool.

□

A second and more complex example can be found in the Appendix section.

Chapter 7

Experimental Evaluation

Motivation In order to empirically evaluate the efficiency of our approach, we developed a large benchmark of Petri nets and their faulty extensions. For this benchmark to be meaningful for practical applications, we analysed a set of real-world examples and abstracted their structure. Then we used the abstracted structure in order to come up with other examples in our benchmark that have a similar structure. We subsequently used the CPN tool to generate the state space of our benchmarks (both for the original Petri nets and their various faulty extensions) and gathered and analysed data in order to answer the following research question:

(RQ3) How scalable is our fault identification technique for unobservable faults?

Answering this research question involves gathering execution times for the Petri nets in our benchmark and performing a regression analysis to develop an empirical model for the efficiency of our approach. By analysing this model we can see how scalable our approach is, i.e., what are the largest sizes of Petri net that can be handled using our approach.

7.1 Experiment design

We studied 10 Petri nets from the practical examples modelled in the CPN toolset with different complexities [38]. We have then used ideas from the field of Complex Networks [60] to characterize the graph structure of those Petri nets. Complex network analysis is a collection of quantitative methods for analysing the structure of a graph and try to come up with characteristics

of such graph. For our analysis we are not going to do an in-depth complex network analysis of Petri net graphs, but we use basic measures inspired by Complex Network Analysis as a starting point. In particular, we abstracted the indegree and outdegree of places and transitions for the above-mentioned 10 examples, and gathered the characterisation data for them.

A visualisation of the gathered data is provided in Figures 7.1 and 7.2. These figures, respectively, show the distribution of the indeegrees and the outdegrees of places and transitions We then calculated the mean value for these measures and subsequently created a controlled benchmark of Petri nets of various sizes respecting these measures.

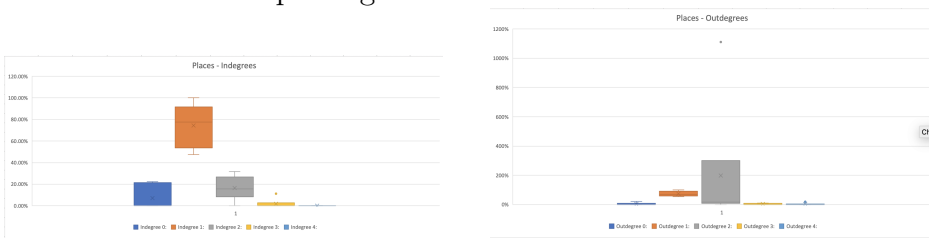

Figure 7.1: The box plot representing indegrees and outdegrees for places.

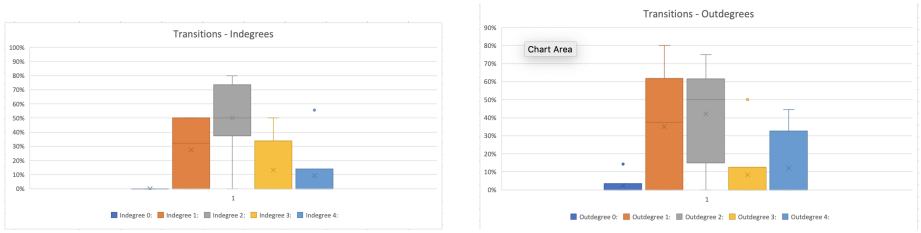

Figure 7.2: The box plot representing indegrees and outdegrees for transitions.

7.2 State space Analysis

In this section, we explain how the state space was created. In general, the state space analysis is useful when we would like to scrutinize behavioural properties of the net, such as boundedness, safeness, liveness, and fairness. For our benchmark, we have used CPN Tools to calculate such a state space.

Before a state space can be calculated and analyzed, we first need to generate the state space code. This can be done by applying the *Enter the state space* tool. To generate the state space code, we need to make sure that

all transitions, places and pages in the net have names. Note that the larger the size of the Petri net is, the more time is needed for the state space to be generated. Once the state space code is generated, we can simply apply the *Calculate state space* tool. The calculated state space can be saved in a report file, when the *Save Report* tool is used. In the report file, statistics section describes the size of the state space and SCC graph.

After entering and calculating the state space, we can display all the nodes contained in it. In order to display the state space nodes we used the *Display state space node* tool. An option for the tool determines which node will be drawn. If the node does not already exist on the page then the node will be displayed. If the node is already on the page, then nothing will be added to the page. Figure 7.3 shows the State Space tools in the CPN toolset.

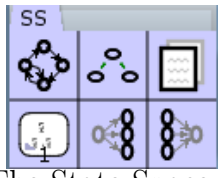

Figure 7.3: The State Space tools in CPN.

The first node created and displayed, represents the initial marking of the net. Then, in order to display the rest of the nodes we need to use the *Display successors* tool to a state space node. The successor nodes to the target node will be displayed, and outgoing arcs from the target node to its successors will be displayed. If any of the successor nodes or outgoing arcs are already displayed and active on the page, then they will not be displayed again. Similarly, *Display predecessors* tool is used to display the outgoing arcs from the predecessors to their target node. Note that the state space was drawn manually, as there was not an option for generating the complete graph at once. For smaller state spaces, it was possible to draw such a graph, but for larger state spaces, it was time consuming as the tool becomes slower as the size of the state space grows.

Figure 7.4 depicts a representation of a state space node. The first number is the number of the node. In this example, the node number is 6. The two numbers at the bottom of the node are the number of predecessors and number of successors that have been calculated. In this case 2 predecessors and 3 successors have been calculated.

Figure 7.4: The graphical representation of a State Space node.

7.3 Examples

In this section, we present a couple of random examples we created according to our methodology and show how we have used them within the CPN Tools.

For all examples, I calculate the same indegree and outdegree measures and only accept those random graphs that deviate in any given measure for at most 20% from the above-specified mean values.

We observe that the larger the size of the Petri net gets, it is easier to generate Petri nets that closely respect these measures, while for a small number of places a small number of deviations can lead to significantly large deviation percentages.

Example 32. The following example illustrates a Petri net with 5 places and 3 transitions, based on corresponding guidelines above.

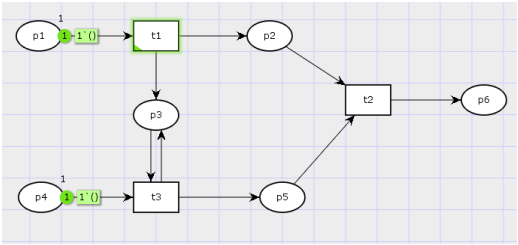

Figure 7.5: A Petri net with 5 places and 3 transitions.

Below is the representation of the generated state space:

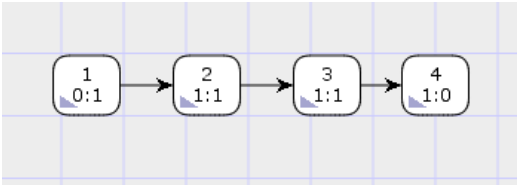

Figure 7.6: The corresponding state space for the Petri net with 5 places and 3 transitions.

Measuring execution time

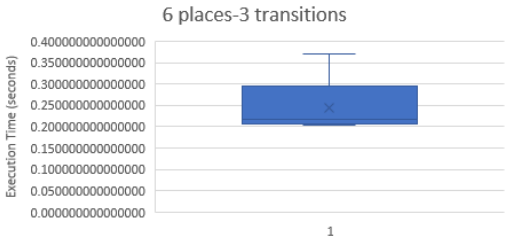

Figure 7.7: The corresponding box plot for the execution time taken for solving such a net using our tool.

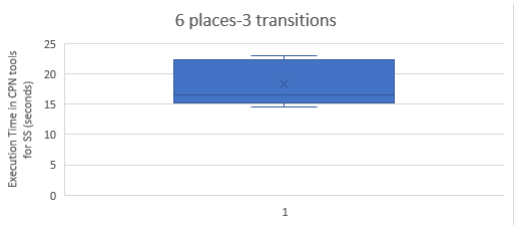

Figure 7.8: The corresponding box plot for the execution time taken for generating the state space in CPN tools.

□

Example 33. The following example illustrates a Petri net with 10 places and 9 transitions, based on corresponding guidelines above.

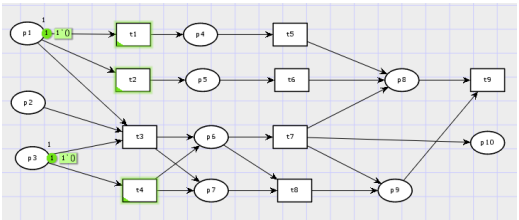

Figure 7.9: A Petri net with 10 places and 9 transitions.

Below is the representation of the generated state space:

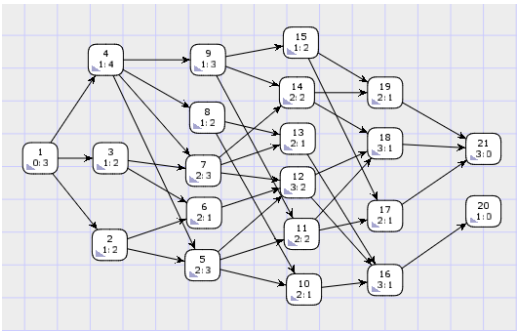

Figure 7.10: The corresponding state space for the Petri net with 10 places and 9 transitions.

Measuring execution time

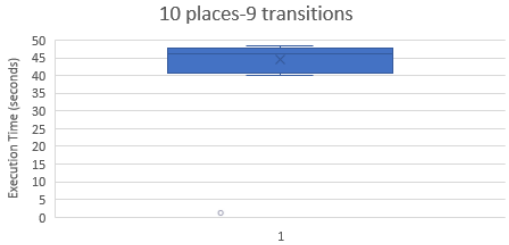

Figure 7.11: The corresponding box plot for the execution time taken for solving such a net using our tool.

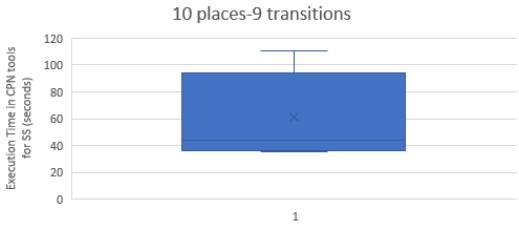

Figure 7.12: The corresponding box plot for the execution time taken for generating the state space in CPN tools.

□

Example 34. The following example illustrates a Petri net with 15 places and 13 transitions, based on corresponding guidelines above.

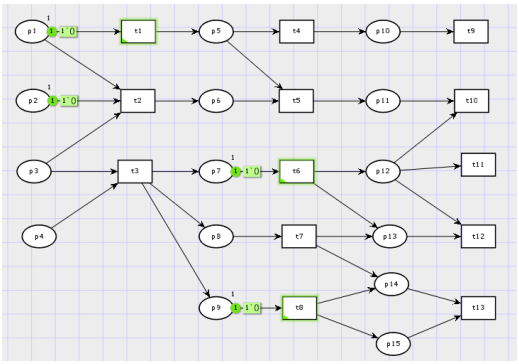

Figure 7.13: A Petri net with 15 places and 13 transitions.

Below is the representation of the generated state space:

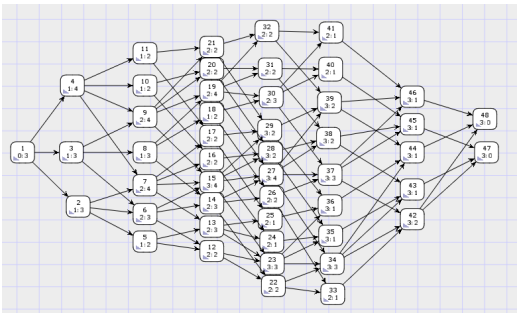

Figure 7.14: The corresponding state space for the Petri net with 15 places and 13 transitions.

Measuring execution time

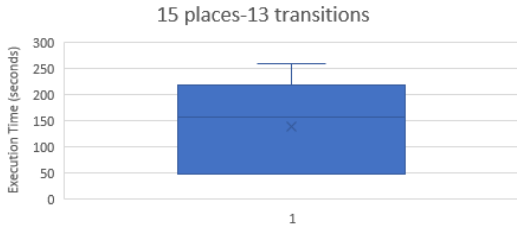

Figure 7.15: The corresponding box plot for the execution time taken for solving such a net using our tool.

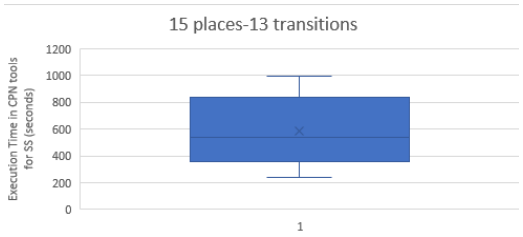

Figure 7.16: The corresponding box plot for the execution time taken for generating the state space in CPN tools.

□

7.4 Experimental results

In this section, I summarize the results of my empirical evaluation. Figure 7.17 depicts a box plot representing the growth of the execution time with respect to the size of the Petri nets (i.e., the number of places; the number of transitions also rise proportionally according to our identified metrics).

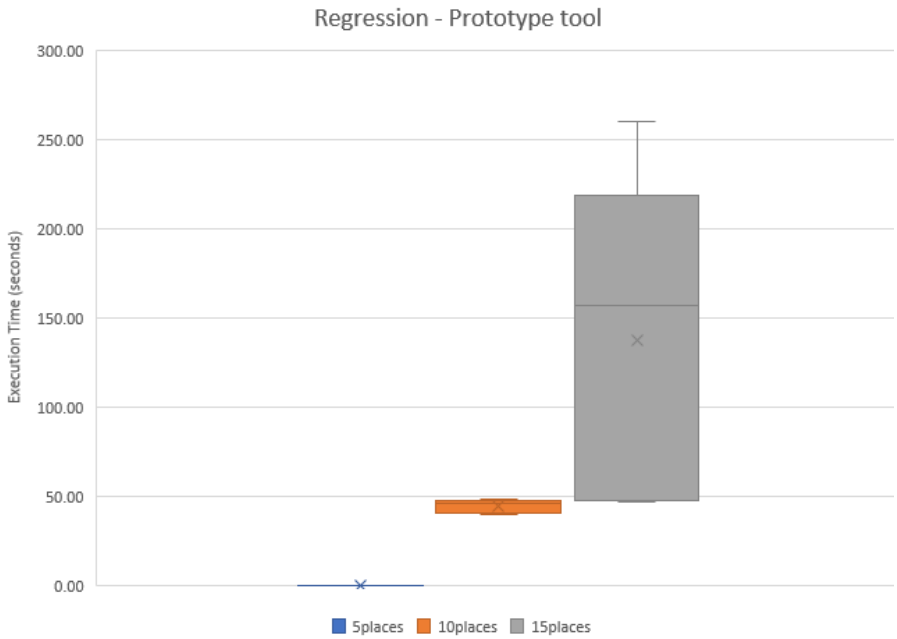

Figure 7.17: Execution time of my tool for different sizes of Petri nets

For comparison, I have also provided the same execution times for the CPN state-space generation in Table 7.18.

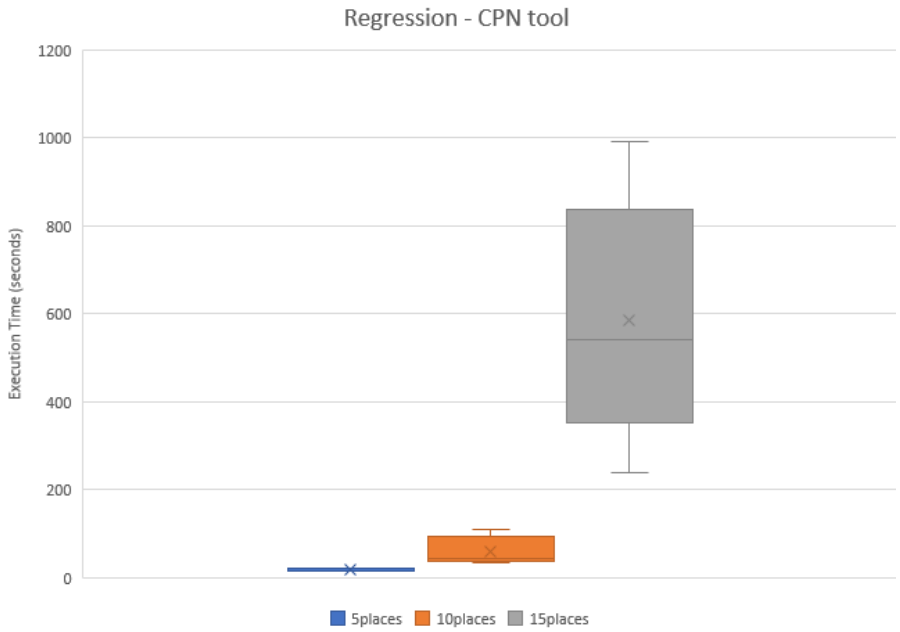

Figure 7.18: Execution time of the CPN tool for different sizes of Petri nets

Final analysis: There is a steep increase in the mean execution time of our tool (from less than a second for 5 places, to 45 seconds for 10 places, to about 150 seconds for 15 places). However, the comparison of this growth with that of state-space generation at CPN tools reveals that our approach is more scalable than explicit-state exploration approaches.

7.5 Threats to validity

In this section, I examine the threat to the validity of our experimental evaluation. I divided these threats into two categories: internal- and external threats to validity.

7.5.1 Internal validity

An internal threat to validity concerns my data set and its analysis method.

The first internal threat to the validity of my results concerns the statistical measures used to generate random subject systems. We can in the future consider more advanced complex network analysis measures to generate random Petri nets that resemble actual cases.

Another threat to internal validity is the limited size of the data set. I have increased the size of the data set to the limits of the computational power of my personal computer. In the future, if I am given access to a more powerful computer / a computing cluster, I could explore the results for larger data set and potentially consider parallelizing the analysis and evaluate the parallelization.

Although we can find different solutions for the fault identification problem, we cannot minimize it to get a unique solution, however results are accurate and valid.

7.5.2 External validity

A threat to external validity is the errors or inaccuracies in the examples used and in the translation between tools (the output of CPN tools and the input of our tool), that may impact our evaluation. To mitigate this, we have carefully scrutinized the examples and the coding both by myself and my supervisor.

Another threat to validity concerns the choice of subject system. To mitigate this threat we have studied already existing examples based on realistic phenomena and we have used ideas from the Complex Networks field to generate other subject systems that have a similar structure. Our study demonstrates that the produced results are largely in agreement across various examples.

Furthermore, our tool was implemented in Python using Z3 SMT Solver to solve the fault identification problem, so the results may not be the same for other programming languages and SMT Solvers. Future studies should validate the generalizability of our findings in other implementation contexts.

7.6 Future work

The tool we have created has not been compared with other tools yet; candidates for such a comparison including synthesis and process mining tools. We are planning to empirically compare the performance of our tool with other

tools using our benchmark examples. Once the tool is tested with a variety of benchmark examples, we will be able to make different comparisons to see if the solutions and the running time are more efficient than the solutions and the running time of the already existing tools.

There is a variety of tools of presenting the format of a Petri net; these include Alpha/Sim, AlPiNa, ARP, Artifex, Cosmos, CPN-AMI, ProM CPN, CPN Tools [39]. These tools are used for the simulation of place/transition nets, high-level Petri nets, stochastic Petri nets, Colored Petri nets, Petri nets with time and so on. In the future we plan to perform an exhaustive research regarding these tools, in order to get an overview of the case studies carried out in these tools and plan an empirical study for our comparison.

Chapter 8

Conclusions

In this chapter, I review the results of this thesis and reflect on the possible avenues for future work.

In the thesis I have formulated the fault identification problem in Petri nets and I have given an answer to the question whether it is possible to efficiently identify unknown faults in a system built based on a known Petri net specification.

I have formalized the basic concepts regarding Petri nets and faults and I have also formalized the problem statement concerning fault identification. I then rephrased a solution scheme which is based on the research done by other authors and in particular by Cabasino. I translated the problem into a system of linear algebraic inequations, taking into consideration two cases, namely when the faults are observable (i.e., faults appear in the net traces) and when the faults are unobservable (i.e., faults do not appear in the net traces). In the case where the faults are unobservable I also make the assumption that the net is loop-free.

I have compared my approach to that of Cabasino and identified some issues with her approach. I have given solutions to these issues and I have shown that my approach holds in those cases that Cabasino's proof have issues.

Last but not least, I have created a tool that automates my approach to fault identification. I have also created a benchmark of examples in order to test my tool. In the thesis I have listed two examples with different complexity in order to show how my tool works and scales to larger examples.

A comparison with other tools has not been done yet, due to the fact that our tool is still a prototype. As future work, I am planning to improve its

performance and test it using my benchmark (as well as any other available set of examples), in order to make a thorough comparison with other tools.

In the future, I plan to polish and finish the generalization of my theorem to multiple faults; this includes different cases, namely, when the faults overlap and when the faults do not overlap. In other words, I plan to relax Assumption 22 and extend the approach to multiple faults. I already have some early results in this direction. I plan to modify the tool in order to automate the extended process (and also validate it on a number of examples) and measure the efficiency of my approach in this extended setting.

Bibliography

[1] Alcaraz-Mejía M, López-Mellado E, Ramírez-Treviño A and River-Rangel I, "Petri Net Based Fault Diagnosis of Discrete Event Systems," *Proceedings of the IEEE*, Mexico, 2003. p. 4730-4735.

[2] Badouel E, Darondeau P, "Petri Net Synthesis," December 2011.

[3] Badouel E, Bernardinello L, Darondeau P, "Petri Net Synthesis," *Texts in theoretical Computer Science, An EATCS series, Springer*, Berlin, Heidelberg, December 2015.

[4] Badouel E, Bernardinello L, Darondeau P, "Polynomial algorithms for the synthesis of bounded nets," *Proceedings of CAAP'95, Lecture Notes in Computer Science, 915*, pp. 647–679, 1995.

[5] Barrett C, "SAT Solvers: Theory and Practice," *Summer School on Verification Technology, Systems and Applications*, New York, USA, September 2008. p. 1-98.

[6] Bergenthum R, Desel J, Lorenz R, Mauser S, "Synthesis of Petri nets from infinite partial languages," *In ACSD08: Proceedings of the 8th International Conference on Application of Concurrency to System Design*, pp. 170–179, Xian, China, 2008.

[7] Bergenthum R, Desel J, Lorenz R, Mauser S, "Process Mining Based on Regions of Languages," *BPM, LNCS 4714*, Heidelberg, Berlin, 2007. p. 375-383.

[8] Bobbio A , "System Modelling with Petri nets," Istituto Elettrotecnico Nazionale Galileo Ferraris, Torino, Italy, 1990. p. 102-143.

[9] Bobbio A, Trivendi K, "System Modelling with Petri nets," Department of Electrical and Computer Engineering, Duke University, Durham, November 2003.

[10] Cabasino M, Giua A, Lafortune S, Seatzu C, "Diagnosability Analysis of Unbounded Petri Nets," *Proceedings of the 48th IEEE International Conference on Decision and Control*, December 2009. p. 1267-1272.

[11] Cabasino M, Giua A, Seatzu C, "Fault detection for discrete event systems using Petri nets with unobservable transitions," *Automatica 46*, May 2010. p. 1531-1539.

[12] Cabasino M, Giua A, Seatzu C, "Identification of Petri Nets from Knowledge of Their Language," *Discrete Event Dyn Syst*, July 2007. p. 447-474.

[13] Cabasino M, Giua A, Hadjicostis C, Seatzu C, "Fault Model Identification with Petri Nets," *Proceedings of the 9th International Workshop on Discrete Event Systems*, Göteborg, Sweden, May 2008. p. 455-461.

[14] Cabasino M, "Diagnosis and Identification of Discrete Event Systems using Petri Nets," Dept. of Electrical and Electronic Engineering, University of Cagliari, Italy, March 2009.

[15] Carmona J, Cortadella J, Kishinevsky M, "A Region-Based Algorithm for Discovering Petri Nets from Event Logs," *In BPM08: Proceedings of the 6th International Conference on Business Process Management*, Heidelberg, Berlin, 2008. p. 358-373.

[16] Carmona J, Cortadella J, Kishinevsky A, Lavagno L, Kondratyev A, Yakovlev A, "A symbolic algorithm for the synthesis of bounded Petri nets," *In ICATPN08: Proceedings International Conferences on Application and Theory of Petri Nets and Other Models of Concurrency*, Berlin, Heidelberg, 2008. p. 92-111.

[17] Carmona J, Cortadella J, Kishinevsky M, "Genet: A Tool for the Synthesis and Mining of Petri Nets," *IEEE: Ninth International Conference on Application of Concurrency to System Design*, Augsburg, Germany, October 2009.

[18] Cortadella J, Kishinevsky M, Lavagno L, Yakovlev A, "Deriving Petri nets from finite transition systems," *IEEE Transactions on Computers, 47(8)*, 1998. p. 859–882.

[19] Darondeau P, "Deriving unbounded Petri nets from formal languages," Institut National de Recherche en Informatique et en Automatique, February, 1998.

[20] Dassow J, Mavlankulov G, Othman M, Turaev S, Selamat H M, Stiebe R,"Grammars Controlled by Petri Nets," August 2012. p. 337-358.

[21] Dotoli M, Fanti P M, Mangini M A, Ukovich W, "On-line fault detection in discrete event systems by Petri nets and integer linear programming," *Automatica 45*, July 2009. p. 2665-2672.

[22] Dotoli M, Fanti P M, Mangini M A, "Fault detection of discrete event systems using Petri nets and integer linear programming," *Proceedings of 17th IFAC World Congress*, Seoul, Korea, July 2008.

[23] Dotoli M, Fanti P M, Mangini M A, "Real time identification of discrete event systems using Petri nets," *Automatica 44*, October 2007. p. 1209-1219.

[24] Ehrenfeucht A, Rozenberg G, " Partial (set) 2-structures. part i,ii," *Acta Informaticag, 27(4)*, 1989. p. 315-368

[25] Esparza J, "Petri Nets, Commutative Context-Free Grammars and Basic Parallel Processes," *Fundamenta Informaticae 30*, 1997. p. 23-41.

[26] Esparza J, Nielsen M, "Decidability Issues for Petri Nets," *BRICS Report Series RS-94-8*, May 1994.

[27] Fanti P M, Seatzu C, "Fault diagnosis and identification of discrete event systems using Petri nets," *Proceedings of the 9th International Workshop on Discrete Event Systems*, Göteborg, Sweden, May 2008. p. 432-435.

[28] Genrich J H, Lautenbach K, "System Modelling with High-Level Petri nets," *Theoretical Computer Science 13*, 1981. p. 109-136.

[29] Giua A, Seatzu C, "A Systems Theory View of Petri nets," *Proceedings of Advances in Control Theory and Applications Series: Lecture Notes in Control and Information Sciences, Vol. 353*, 2007.

[30] Giua A, Seatzu C, "Identification of free-labelled Petri nets via integer programming," *Proceedings of the 44th IEEE Conference on Decision and Control*, Spain, December 2005.

[31] Giua A, "Petri Nets as Discrete Event Models for Supervisory Control," Rensselaer Polytechnic Institute, New York, July 1992.

[32] Gomes P C, Kautz H, Sabharwal A, Selman B, "Chapter 2: Satisfiability Solvers," *Handbook of Knowledge Representation*, 2008. p. 89-134.

[33] Hack M, "Decidability Questions for Petri Nets," Massachusetts Institute of Technology, December 1975.

[34] Hack M, "Petri Net Languages," Massachusetts Institute of Technology, March 1976.

[35] Hiraishi K,"Construction of a Class of Safe Petri Nets by Presenting Firing Sequences," *Proceedings of the 13th IEEE International Conference on Application and Theory of Petri Nets, Vol. 616*, United Kingdom, June 1992. p. 244-262.

[36] Hořeňovský M, "Modern SAT solvers: fast, neat and underused ," August 2018.

[37] Jantzen M, "Language Theory of Petri Nets," Fachbereich Informatik Universitat Hamburg, 1987. p. 398-412.

[38] Jensen K, Christensen S, Kristensen L. M., Westergaard M, Verbeek H.M.W. (Eric), January 2018, CPN Tools - Sample CPN Models, Eindhoven University of Technology, viewed 17 July 2020, http://cpntools.org/2018/01/08/sample-cpn-models.

[39] Jie W T, Ameedeen A M, "A Survey of Petri Net Tools," *Article in Lecture Notes in Electrical Engineering*, January, 2015.

[40] Kiehn A, "Petri Net Systems and their Closure Properties," *Proceedings of Advances in Petri Nets in the 9th European Workshop on Applications and Theory in Petri Nets*, June 1988.

[41] Koutny M, Pietkiewicz-Koutny M, "Synthesis of General Petri Nets with Localities," School of Computing Science, Newcastle, United Kingdom, 2009.

[42] Li L, Hadjicostis C, Sreenivas S R, "Fault Detection and Identification in Petri Net Controllers," *Proceedings of the 43rd IEEE International Conference on Decision and Control*, December 2004. p. 5248-5253.

[43] Lorenz R, Bergenthum R, Desel J, Mauser S, "Synthesis of Petri nets from finite partial languages," *In ACSD07: Proceedings of the 7th International Conference on Application of Con- currency to System Design*, Washington, DC, USA, 2007. p. 157–166.

[44] Lorenz R, Juhas G, "Towards synthesis of Petri nets from scenarios," *In Proc. of 27th Inter- national Conference on Applications and Theory of Petri Nets and Other Models of Concurrency*, 2006. p. 302–321.

[45] Lorenz R, Juhas G, Mauser S, "How to synthesize nets from languages – a survey," *In Proc. 2007 Winter Simulation Conference*,Washington DC, USA, December 2007.

[46] Murata T, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE, VOL. 77, NO. 4*, April 1989. p. 541-580.

[47] Pelz E, "Closure Properties of Deterministic Petri Nets," Université Paris Sud, France, 1987. p. 371-382.

[48] Peterson J L, "Petri Nets," *Computing Surveys, Vol 9, No. 3*, September 1977. p. 223-252.

[49] Petri A C, "Kommunication mit Automaten," PhD thesis, Institut fur Instrumentelle Mathematik, Schriften des IIM, No. 3, Bonn, Germany, 1962.

[50] Prock J,"A New Technique for Fault Detection Using Petri Nets," *Automatica, Vol. 27, No. 2*, 1991. p. 239-245.

[51] Purvis M, "Dynamic Modelling of Legal Processes with Petri Nets," University of Otago, Dunedin, New Zealand, February 1998.

[52] Srinivasan S V, Jafari A M , "Fault Detection/Monitoring Using Time Petri Nets," *Proceedings of the IEEE, Transactions on Systems, Man, and Cybernetics, VOL. 23, NO 4*, August 1993. p. 1155-1162.

[53] Starke H P, "Free Petri Net Languages," Sektion Mathematik der Humboldt-Univ., Berlin, 1978. p. 506-515.

[54] Thomas W, "Applied Automata Theory," RWTH Aachen University, November 2005.

[55] Van der Aalst W M P, Günthe C, "Finding structure in unstructered processes: The case of process mining," *In Proceedings of ACSD 2007, invited paper*, 2007.

[56] Van der Aalst W M P, Van Dongen B F, Herbst J, Maruster L, Schimm G, Weijters A J M M, "Workflow mining: A survey of issues and approaches," *Data Knowl. Eng. 47 (2)*, 2003. p. 237-267.

[57] Van der Aalst W M P, Weijters A J M M, Maruster L, "Workflow mining: Discovering process models from event logs," *IEEE Trans. on Data & Knowledge Engineering, 16(9)*, 2004. p. 128–1142.

[58] Van der Aalst W M P, Weijters A J M M, "Process Mining, Discovering Workflow Models from Event-Based Data," 2004.

[59] Van der Aalst W M P, "Decomposing Petri nets for process mining: A generic approach," *Distrib Parallel Databases 31*, July 2013. p. 471-507.

[60] Van Steen M, "Graph Theory and Complex Networks," *ISBN: 978-90-815406-1-2*, April 2012.

[61] Wang J,"Petri Nets for Dynamic Event-Driven System Modeling," *CRC Press LLC*, Monmouth University, 2002. p. 24.1-24.19.

[62] Wasbo O S, Foss A B, "Modelling unit processes using formal language description and object-orientation," *Proceedings of the Mathematical Modelling of Systems, Vol. 1, No. 1*, 1996. p. 000-111.

[63] Wu Y, Hadjicostis C, "Algebraic Approaches for Fault Identification in Discrete-Event Systems," *Proceedings of the IEEE Transactions on Automatic Control, VOL. 50, NO. 12*, December 2005. p. 2048-2064.

[64] Yau S S, Caglayan U M, "Distributed Software System Design Representation Using Modified Petri Nets," *Proceedings of the IEEE Transactions on Software Engineering, VOL. SE-9, NO. 6*, November 1983. p. 733-745.

Appendix A

Implementation Test Case

In this appendix, we provide the details of another test case created for my prototype implementation. I starting from the graphical representation of the original net system and then present its non-faulty and faulty languages. Then I present the generated Python file for the example and finally, present and discuss the outcome of the solution that is automatically generated by the prototype tool.

Example 35. Let N be the specification Petri net in Figure A.1.

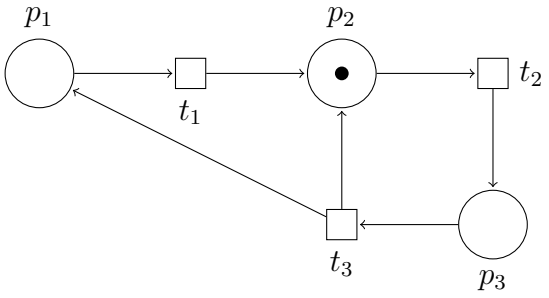

Figure A.1: The fault-free net system.

Hence, the initial marking of N is $\mu_0 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ (as shown).

The Pre and $Post$ conditions of the Petri net N are

$$Pre = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, Post = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Let $L_{\leq q}(N) = \{\lambda, t_2, t_2t_3, t_2t_3t_1, t_2t_3t_2, t_2t_3t_1t_2, t_2t_3t_2t_1, t_2t_3t_2t_3\}$ and $L^f_{\leq q} = \{\lambda, t_2, t_2t_3, t_2t_3t_1, t_2t_3t_2, t_2t_3t_1t_2, t_2t_3t_2t_1, t_2t_3t_2t_3, t_1, t_1t_1, t_1t_2, t_1t_1t_1, t_1t_1t_2, t_1t_2t_3, t_2t_3t_1t_1, t_1t_2t_3t_1, t_1t_2t_3t_2, t_1t_1t_1t_1, t_1t_1t_1t_2\}$, where q=4.

Also let the sequences $\notin L^f = \{t_3, t_1t_3, t_2t_1, t_2t_2, t_1t_1t_3, t_1t_2t_1, t_1t_2t_2, t_2t_3t_3, t_2t_3t_1t_3, t_2t_3t_2t_2, t_1t_2t_3t_3, t_1t_1t_1t_3\}$

The input file to our tool, based on our syntax, is presented in Figure A.2.

```
places
3
p1
p2
p3
transitions
3
t1
t2
t3
mu0
0
1
0
pre
1 0 0
0 1 0
0 0 1
post
0 0 1
1 0 1
0 1 0
in_Lf(18)
t1
[0,0,0]
t2
[0,0,0]
t1,t1
[1,0,0]
t1,t2
[1,0,0]
t2,t3
[0,1,0]
t1,t1,t1
[2,0,0]
t1,t1,t2
[2,0,0]
```

```
t1,t2,t3
[1,1,0]
t2,t3,t1
[0,1,1]
t2,t3,t2
[0,1,1]
t2,t3,t1,t2
[1,1,1]
t2,t3,t1,t1
[1,1,1]
t2,t3,t2,t1
[0,2,1]
t2,t3,t2,t3
[0,2,1]
t1,t2,t3,t1
[1,1,1]
t1,t2,t3,t2
[1,1,1]
t1,t1,t1,t1
[3,0,0]
t1,t1,t1,t2
[3,0,0]
not_in_Lf(12)
t3
[0,0,0]
t1,t3
[1,0,0]
t2,t1
[0,1,0]
t2,t2
[0,1,0]
t1,t1,t3
[2,0,0]
t1,t2,t1
[1,1,0]
t1,t2,t2
[1,1,0]
t2,t3,t3
[0,1,1]
t2,t3,t2,t2
[0,2,1]
t2,t3,t1,t3
[1,1,1]
t1,t1,t1,t3
[3,0,0]
```

```
t1,t2,t3,t3
[1,1,1]
```

Figure A.2: The index file based on the fault-free net.

We can solve this problem by using the sets

$PL(N)_{\leq q} = \{(\epsilon, t_2), (t_2, t_3), (t_2t_3, t_1), (t_2t_3, t_2), (t_2t_3t_1, t_2), (t_2t_3t_2, t_1), (t_2t_3t_2, t_3)\}$

$PL(N)^f_{\leq q} = \{(\epsilon, t_1), (t_1, t_1), (t_1, t_2), (t_1t_1, t_1), (t_1t_1, t_2), (t_1t_2, t_3), (t_2t_3t_1, t_1), (t_1t_2t_3, t_1), (t_1t_2t_3, t_2), (t_1t_1t_1, t_1), (t_1t_1t_1, t_2)\}$

and

$PNL(N)^f_{\leq q} = \{(t_3), (t_1, t_3), (t_2, t_1), (t_2, t_2), (t_1t_1, t_3), (t_1t_2, t_1), (t_1t_2, t_2), (t_2t_3, t_3), (t_2t_3t_1, t_3), (t_2t_3t_2, t_2), (t_1t_2t_3, t_3), (t_1t_1t_1, t_3)\}$

The generated file is as follows

```
#!/bin/python

from z3 import *

# we have that
s = Solver()
## mu0_px is the initial marking for place px;
mu_p1, mu_p2, mu_p3 = 0, 1, 0

## pi_tj is the pre-condition from place pi to transition tj
p1_t1, p1_t2, p1_t3 = 1, 0, 0
p2_t1, p2_t2, p2_t3 = 0, 1, 0
p3_t1, p3_t2, p3_t3 = 0, 0, 1

## tj_pi is the post-condition from transition tj to place pi
t1_p1, t2_p1, t3_p1 = 0, 0, 1
t1_p2, t2_p2, t3_p2 = 1, 0, 1
t1_p3, t2_p3, t3_p3 = 0, 1, 0

## find the values for the faulty transitions
f_p1, p1_f = Ints('f_p1 p1_f')
f_p2, p2_f = Ints('f_p2 p2_f')
f_p3, p3_f = Ints('f_p3 p3_f')

# where they should be
s.add( f_p1 >= 0, f_p2 >= 0, f_p3 >= 0 )
s.add( p1_f >= 0, p2_f >= 0, p3_f >= 0 )

s.add(And(
```

```
    Or( And((f_p1 >= 0, p1_f == 0)),
        And((f_p1 == 0, p1_f >= 0)),
        And((f_p1 == 0, p1_f == 0))),
    Or( And((f_p2 >= 0, p2_f == 0)),
        And((f_p2 == 0, p2_f >= 0)),
        And((f_p2 == 0, p2_f == 0))),
    Or( And((f_p3 >= 0, p3_f == 0)),
        And((f_p3 == 0, p3_f >= 0)),
        And((f_p3 == 0, p3_f == 0)))
     )
)

## l \in Naturals ;
l0 = Int('l0')
l1 = Int('l1')
l2 = Int('l2')
l3 = Int('l3')
l4 = Int('l4')
l5 = Int('l5')
l6 = Int('l6')
l7 = Int('l7')
l8 = Int('l8')
l9 = Int('l9')
l10 = Int('l10')
l11 = Int('l11')
l12 = Int('l12')
l13 = Int('l13')
l14 = Int('l14')
l15 = Int('l15')
l16 = Int('l16')
l17 = Int('l17')
l18 = Int('l18')
l19 = Int('l19')
l20 = Int('l20')
l21 = Int('l21')
l22 = Int('l22')
l23 = Int('l23')
l24 = Int('l24')
l25 = Int('l25')
l26 = Int('l26')
l27 = Int('l27')
l28 = Int('l28')
l29 = Int('l29')

########################################
```

```
## \in L^f (Equation 4.1)
########################################
# Sequence 0: t1
s0_t1, s0_t2, s0_t3 = 0, 0, 0
s.add(
   Exists([l0],
      And( Implies(l0 >= 0,
      And(l0 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s0_t1 + (t2_p1-p1_t2)*s0_t2
         + (t3_p1-p1_t3)*s0_t3 + l0 * (f_p1 - p1_f) >= p1_t1,
         mu_p2 + (t1_p2-p2_t1)*s0_t1 + (t2_p2-p2_t2)*s0_t2
         + (t3_p2-p2_t3)*s0_t3 + l0 * (f_p2 - p2_f) >= p2_t1,
         mu_p3 + (t1_p3-p3_t1)*s0_t1 + (t2_p3-p3_t2)*s0_t2
         + (t3_p3-p3_t3)*s0_t3 + l0 * (f_p3 - p3_f) >= p3_t1,
)))))

# Sequence 1: t2
s1_t1, s1_t2, s1_t3 = 0, 0, 0
s.add(
   Exists([l1],
      And( Implies(l1 >= 0,
      And(l1 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s1_t1 + (t2_p1-p1_t2)*s1_t2
         + (t3_p1-p1_t3)*s1_t3 + l1 * (f_p1 - p1_f) >= p1_t2,
         mu_p2 + (t1_p2-p2_t1)*s1_t1 + (t2_p2-p2_t2)*s1_t2
         + (t3_p2-p2_t3)*s1_t3 + l1 * (f_p2 - p2_f) >= p2_t2,
         mu_p3 + (t1_p3-p3_t1)*s1_t1 + (t2_p3-p3_t2)*s1_t2
         + (t3_p3-p3_t3)*s1_t3 + l1 * (f_p3 - p3_f) >= p3_t2,
)))))

# Sequence 2: t1,t1
s2_t1, s2_t2, s2_t3 = 1, 0, 0
s.add(
   Exists([l2],
      And( Implies(l2 >= 0,
      And(l2 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s2_t1 + (t2_p1-p1_t2)*s2_t2
         + (t3_p1-p1_t3)*s2_t3 + l2 * (f_p1 - p1_f) >= p1_t1,
         mu_p2 + (t1_p2-p2_t1)*s2_t1 + (t2_p2-p2_t2)*s2_t2
         + (t3_p2-p2_t3)*s2_t3 + l2 * (f_p2 - p2_f) >= p2_t1,
         mu_p3 + (t1_p3-p3_t1)*s2_t1 + (t2_p3-p3_t2)*s2_t2
         + (t3_p3-p3_t3)*s2_t3 + l2 * (f_p3 - p3_f) >= p3_t1,
)))))

# Sequence 3: t1,t2
```

```
s3_t1, s3_t2, s3_t3 = 1, 0, 0
s.add(
   Exists([l3],
      And( Implies(l3 >= 0,
      And(l3 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s3_t1 + (t2_p1-p1_t2)*s3_t2
         + (t3_p1-p1_t3)*s3_t3 + l3 * (f_p1 - p1_f) >= p1_t2,
         mu_p2 + (t1_p2-p2_t1)*s3_t1 + (t2_p2-p2_t2)*s3_t2
         + (t3_p2-p2_t3)*s3_t3 + l3 * (f_p2 - p2_f) >= p2_t2,
         mu_p3 + (t1_p3-p3_t1)*s3_t1 + (t2_p3-p3_t2)*s3_t2
         + (t3_p3-p3_t3)*s3_t3 + l3 * (f_p3 - p3_f) >= p3_t2,
)))))

# Sequence 4: t2,t3
s4_t1, s4_t2, s4_t3 = 0, 1, 0
s.add(
   Exists([l4],
      And( Implies(l4 >= 0,
      And(l4 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s4_t1 + (t2_p1-p1_t2)*s4_t2
         + (t3_p1-p1_t3)*s4_t3 + l4 * (f_p1 - p1_f) >= p1_t3,
         mu_p2 + (t1_p2-p2_t1)*s4_t1 + (t2_p2-p2_t2)*s4_t2
         + (t3_p2-p2_t3)*s4_t3 + l4 * (f_p2 - p2_f) >= p2_t3,
         mu_p3 + (t1_p3-p3_t1)*s4_t1 + (t2_p3-p3_t2)*s4_t2
         + (t3_p3-p3_t3)*s4_t3 + l4 * (f_p3 - p3_f) >= p3_t3,
)))))

# Sequence 5: t1,t1,t1
s5_t1, s5_t2, s5_t3 = 2, 0, 0
s.add(
   Exists([l5],
      And( Implies(l5 >= 0,
      And(l5 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s5_t1 + (t2_p1-p1_t2)*s5_t2
         + (t3_p1-p1_t3)*s5_t3 + l5 * (f_p1 - p1_f) >= p1_t1,
         mu_p2 + (t1_p2-p2_t1)*s5_t1 + (t2_p2-p2_t2)*s5_t2
         + (t3_p2-p2_t3)*s5_t3 + l5 * (f_p2 - p2_f) >= p2_t1,
         mu_p3 + (t1_p3-p3_t1)*s5_t1 + (t2_p3-p3_t2)*s5_t2
         + (t3_p3-p3_t3)*s5_t3 + l5 * (f_p3 - p3_f) >= p3_t1,
)))))

# Sequence 6: t1,t1,t2
s6_t1, s6_t2, s6_t3 = 2, 0, 0
s.add(
   Exists([l6],
```

```
      And( Implies(l6 >= 0,
      And(l6 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s6_t1 + (t2_p1-p1_t2)*s6_t2
         + (t3_p1-p1_t3)*s6_t3 + l6 * (f_p1 - p1_f) >= p1_t2,
         mu_p2 + (t1_p2-p2_t1)*s6_t1 + (t2_p2-p2_t2)*s6_t2
         + (t3_p2-p2_t3)*s6_t3 + l6 * (f_p2 - p2_f) >= p2_t2,
         mu_p3 + (t1_p3-p3_t1)*s6_t1 + (t2_p3-p3_t2)*s6_t2
         + (t3_p3-p3_t3)*s6_t3 + l6 * (f_p3 - p3_f) >= p3_t2,
)))))

# Sequence 7: t1,t2,t3
s7_t1, s7_t2, s7_t3 = 1, 1, 0
s.add(
   Exists([l7],
      And( Implies(l7 >= 0,
      And(l7 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s7_t1 + (t2_p1-p1_t2)*s7_t2
         + (t3_p1-p1_t3)*s7_t3 + l7 * (f_p1 - p1_f) >= p1_t3,
         mu_p2 + (t1_p2-p2_t1)*s7_t1 + (t2_p2-p2_t2)*s7_t2
         + (t3_p2-p2_t3)*s7_t3 + l7 * (f_p2 - p2_f) >= p2_t3,
         mu_p3 + (t1_p3-p3_t1)*s7_t1 + (t2_p3-p3_t2)*s7_t2
         + (t3_p3-p3_t3)*s7_t3 + l7 * (f_p3 - p3_f) >= p3_t3,
)))))

# Sequence 8: t2,t3,t1
s8_t1, s8_t2, s8_t3 = 0, 1, 1
s.add(
   Exists([l8],
      And( Implies(l8 >= 0,
      And(l8 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s8_t1 + (t2_p1-p1_t2)*s8_t2
         + (t3_p1-p1_t3)*s8_t3 + l8 * (f_p1 - p1_f) >= p1_t1,
         mu_p2 + (t1_p2-p2_t1)*s8_t1 + (t2_p2-p2_t2)*s8_t2
         + (t3_p2-p2_t3)*s8_t3 + l8 * (f_p2 - p2_f) >= p2_t1,
         mu_p3 + (t1_p3-p3_t1)*s8_t1 + (t2_p3-p3_t2)*s8_t2
         + (t3_p3-p3_t3)*s8_t3 + l8 * (f_p3 - p3_f) >= p3_t1,
)))))

# Sequence 9: t2,t3,t2
s9_t1, s9_t2, s9_t3 = 0, 1, 1
s.add(
   Exists([l9],
      And( Implies(l9 >= 0,
      And(l9 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s9_t1 + (t2_p1-p1_t2)*s9_t2
```

```
         + (t3_p1-p1_t3)*s9_t3 + l9 * (f_p1 - p1_f) >= p1_t2,
         mu_p2 + (t1_p2-p2_t1)*s9_t1 + (t2_p2-p2_t2)*s9_t2
         + (t3_p2-p2_t3)*s9_t3 + l9 * (f_p2 - p2_f) >= p2_t2,
         mu_p3 + (t1_p3-p3_t1)*s9_t1 + (t2_p3-p3_t2)*s9_t2
         + (t3_p3-p3_t3)*s9_t3 + l9 * (f_p3 - p3_f) >= p3_t2,
)))))

# Sequence 10: t2,t3,t1,t2
s10_t1, s10_t2, s10_t3 = 1, 1, 1
s.add(
   Exists([l10],
      And( Implies(l10 >= 0,
      And(l10 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s10_t1 + (t2_p1-p1_t2)*s10_t2
         + (t3_p1-p1_t3)*s10_t3 + l10 * (f_p1 - p1_f)>=p1_t2,
         mu_p2 + (t1_p2-p2_t1)*s10_t1 + (t2_p2-p2_t2)*s10_t2
         + (t3_p2-p2_t3)*s10_t3 + l10 * (f_p2 - p2_f)>=p2_t2,
         mu_p3 + (t1_p3-p3_t1)*s10_t1 + (t2_p3-p3_t2)*s10_t2
         + (t3_p3-p3_t3)*s10_t3 + l10 * (f_p3 - p3_f)>=p3_t2,
)))))

# Sequence 11: t2,t3,t1,t1
s11_t1, s11_t2, s11_t3 = 1, 1, 1
s.add(
   Exists([l11],
      And( Implies(l11 >= 0,
      And(l11 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s11_t1 + (t2_p1-p1_t2)*s11_t2
         + (t3_p1-p1_t3)*s11_t3 + l11 * (f_p1 - p1_f)>=p1_t1,
         mu_p2 + (t1_p2-p2_t1)*s11_t1 + (t2_p2-p2_t2)*s11_t2
         + (t3_p2-p2_t3)*s11_t3 + l11 * (f_p2 - p2_f)>=p2_t1,
         mu_p3 + (t1_p3-p3_t1)*s11_t1 + (t2_p3-p3_t2)*s11_t2
         + (t3_p3-p3_t3)*s11_t3 + l11 * (f_p3 - p3_f)>=p3_t1,
)))))

# Sequence 12: t2,t3,t2,t1
s12_t1, s12_t2, s12_t3 = 0, 2, 1
s.add(
   Exists([l12],
      And( Implies(l12 >= 0,
      And(l12 >= 0,
         mu_p1 + (t1_p1-p1_t1)*s12_t1 + (t2_p1-p1_t2)*s12_t2
         + (t3_p1-p1_t3)*s12_t3 + l12 * (f_p1 - p1_f)>=p1_t1,
         mu_p2 + (t1_p2-p2_t1)*s12_t1 + (t2_p2-p2_t2)*s12_t2
         + (t3_p2-p2_t3)*s12_t3 + l12 * (f_p2 - p2_f)>=p2_t1,
```

```
            mu_p3 + (t1_p3-p3_t1)*s12_t1 + (t2_p3-p3_t2)*s12_t2
            + (t3_p3-p3_t3)*s12_t3 + l12 * (f_p3 - p3_f)>=p3_t1,
)))))

# Sequence 13: t2,t3,t2,t3
s13_t1, s13_t2, s13_t3 = 0, 2, 1
s.add(
   Exists([l13],
      And( Implies(l13 >= 0,
      And(l13 >= 0,
            mu_p1 + (t1_p1-p1_t1)*s13_t1 + (t2_p1-p1_t2)*s13_t2
            + (t3_p1-p1_t3)*s13_t3 + l13 * (f_p1 - p1_f)>=p1_t3,
            mu_p2 + (t1_p2-p2_t1)*s13_t1 + (t2_p2-p2_t2)*s13_t2
            + (t3_p2-p2_t3)*s13_t3 + l13 * (f_p2 - p2_f)>=p2_t3,
            mu_p3 + (t1_p3-p3_t1)*s13_t1 + (t2_p3-p3_t2)*s13_t2
            + (t3_p3-p3_t3)*s13_t3 + l13 * (f_p3 - p3_f)>=p3_t3,
)))))

# Sequence 14: t1,t2,t3,t1
s14_t1, s14_t2, s14_t3 = 1, 1, 1
s.add(
   Exists([l14],
      And( Implies(l14 >= 0,
      And(l14 >= 0,
            mu_p1 + (t1_p1-p1_t1)*s14_t1 + (t2_p1-p1_t2)*s14_t2
            + (t3_p1-p1_t3)*s14_t3 + l14 * (f_p1 - p1_f)>=p1_t1,
            mu_p2 + (t1_p2-p2_t1)*s14_t1 + (t2_p2-p2_t2)*s14_t2
            + (t3_p2-p2_t3)*s14_t3 + l14 * (f_p2 - p2_f)>=p2_t1,
            mu_p3 + (t1_p3-p3_t1)*s14_t1 + (t2_p3-p3_t2)*s14_t2
            + (t3_p3-p3_t3)*s14_t3 + l14 * (f_p3 - p3_f)>=p3_t1,
)))))

# Sequence 15: t1,t2,t3,t2
s15_t1, s15_t2, s15_t3 = 1, 1, 1
s.add(
   Exists([l15],
      And( Implies(l15 >= 0,
      And(l15 >= 0,
            mu_p1 + (t1_p1-p1_t1)*s15_t1 + (t2_p1-p1_t2)*s15_t2
            + (t3_p1-p1_t3)*s15_t3 + l15 * (f_p1 - p1_f)>=p1_t2,
            mu_p2 + (t1_p2-p2_t1)*s15_t1 + (t2_p2-p2_t2)*s15_t2
            + (t3_p2-p2_t3)*s15_t3 + l15 * (f_p2 - p2_f)>=p2_t2,
            mu_p3 + (t1_p3-p3_t1)*s15_t1 + (t2_p3-p3_t2)*s15_t2
            + (t3_p3-p3_t3)*s15_t3 + l15 * (f_p3 - p3_f)>=p3_t2,
)))))
```

```
300
301 # Sequence 16: t1,t1,t1,t1
302 s16_t1, s16_t2, s16_t3 = 3, 0, 0
303 s.add(
304    Exists([l16],
305       And( Implies(l16 >= 0,
306       And(l16 >= 0,
307          mu_p1 + (t1_p1-p1_t1)*s16_t1 + (t2_p1-p1_t2)*s16_t2
308          + (t3_p1-p1_t3)*s16_t3 + l16 * (f_p1 - p1_f)>=p1_t1,
309          mu_p2 + (t1_p2-p2_t1)*s16_t1 + (t2_p2-p2_t2)*s16_t2
310          + (t3_p2-p2_t3)*s16_t3 + l16 * (f_p2 - p2_f)>=p2_t1,
311          mu_p3 + (t1_p3-p3_t1)*s16_t1 + (t2_p3-p3_t2)*s16_t2
312          + (t3_p3-p3_t3)*s16_t3 + l16 * (f_p3 - p3_f)>=p3_t1,
313 )))))
314
315 # Sequence 17: t1,t1,t1,t2
316 s17_t1, s17_t2, s17_t3 = 3, 0, 0
317 s.add(
318    Exists([l17],
319       And( Implies(l17 >= 0,
320       And(l17 >= 0,
321          mu_p1 + (t1_p1-p1_t1)*s17_t1 + (t2_p1-p1_t2)*s17_t2
322          + (t3_p1-p1_t3)*s17_t3 + l17 * (f_p1 - p1_f)>=p1_t2,
323          mu_p2 + (t1_p2-p2_t1)*s17_t1 + (t2_p2-p2_t2)*s17_t2
324          + (t3_p2-p2_t3)*s17_t3 + l17 * (f_p2 - p2_f)>=p2_t2,
325          mu_p3 + (t1_p3-p3_t1)*s17_t1 + (t2_p3-p3_t2)*s17_t2
326          + (t3_p3-p3_t3)*s17_t3 + l17 * (f_p3 - p3_f)>=p3_t2,
327 )))))
328
329 ########################################
330 ## \not \in L^f (Equation 4.2)
331 ########################################
332 # Sequence 18: t3
333 s18_t1, s18_t2, s18_t3 = 0, 0, 0
334 s.add(
335    ForAll([l18],
336       And( Implies(l18 >= 0,
337       Or(
338          And(l18 >= 0,  mu_p1 + (t1_p1-p1_t1)*s18_t1
339          + (t2_p1-p1_t2)*s18_t2 + (t3_p1-p1_t3)*s18_t3
340          + l18 * (f_p1 - p1_f) < p1_t3),
341          And(l18 >= 0,  mu_p2 + (t1_p2-p2_t1)*s18_t1
342          + (t2_p2-p2_t2)*s18_t2 + (t3_p2-p2_t3)*s18_t3
343          + l18 * (f_p2 - p2_f) < p2_t3),
344          And(l18 >= 0,  mu_p3 + (t1_p3-p3_t1)*s18_t1
```

```
345          + (t2_p3-p3_t2)*s18_t2 + (t3_p3-p3_t3)*s18_t3
346          + l18 * (f_p3 - p3_f) < p3_t3),
347 )))))
348
349 # Sequence 19: f,t1,t3
350 s19_t1, s19_t2, s19_t3 = 1, 0, 0
351 s.add(
352    ForAll([l19],
353       And( Implies(l19 >= 1,
354       Or(
355          And(l19 >= 1,  mu_p1 + (t1_p1-p1_t1)*s19_t1
356          + (t2_p1-p1_t2)*s19_t2 + (t3_p1-p1_t3)*s19_t3
357          + l19 * (f_p1 - p1_f) < p1_t3),
358          And(l19 >= 1,  mu_p2 + (t1_p2-p2_t1)*s19_t1
359          + (t2_p2-p2_t2)*s19_t2 + (t3_p2-p2_t3)*s19_t3
360          + l19 * (f_p2 - p2_f) < p2_t3),
361          And(l19 >= 1,  mu_p3 + (t1_p3-p3_t1)*s19_t1
362          + (t2_p3-p3_t2)*s19_t2 + (t3_p3-p3_t3)*s19_t3
363          + l19 * (f_p3 - p3_f) < p3_t3),
364 )))))
365
366 # Sequence 20: t2,t1
367 s20_t1, s20_t2, s20_t3 = 0, 1, 0
368 s.add(
369    ForAll([l20],
370       And( Implies(l20 >= 0,
371       Or(
372          And(l20 >= 0,  mu_p1 + (t1_p1-p1_t1)*s20_t1
373          + (t2_p1-p1_t2)*s20_t2 + (t3_p1-p1_t3)*s20_t3
374          + l20 * (f_p1 - p1_f) < p1_t1),
375          And(l20 >= 0,  mu_p2 + (t1_p2-p2_t1)*s20_t1
376          + (t2_p2-p2_t2)*s20_t2 + (t3_p2-p2_t3)*s20_t3
377          + l20 * (f_p2 - p2_f) < p2_t1),
378          And(l20 >= 0,  mu_p3 + (t1_p3-p3_t1)*s20_t1
379          + (t2_p3-p3_t2)*s20_t2 + (t3_p3-p3_t3)*s20_t3
380          + l20 * (f_p3 - p3_f) < p3_t1),
381 )))))
382
383 # Sequence 21: t2,t2
384 s21_t1, s21_t2, s21_t3 = 0, 1, 0
385 s.add(
386    ForAll([l21],
387       And( Implies(l21 >= 0,
388       Or(
389          And(l21 >= 0,  mu_p1 + (t1_p1-p1_t1)*s21_t1
```

```
         + (t2_p1-p1_t2)*s21_t2 + (t3_p1-p1_t3)*s21_t3
         + l21 * (f_p1 - p1_f) < p1_t2),
         And(l21 >= 0,  mu_p2 + (t1_p2-p2_t1)*s21_t1
         + (t2_p2-p2_t2)*s21_t2 + (t3_p2-p2_t3)*s21_t3
         + l21 * (f_p2 - p2_f) < p2_t2),
         And(l21 >= 0,  mu_p3 + (t1_p3-p3_t1)*s21_t1
         + (t2_p3-p3_t2)*s21_t2 + (t3_p3-p3_t3)*s21_t3
         + l21 * (f_p3 - p3_f) < p3_t2),
)))))

# Sequence 22: f,t1,t1,t3
s22_t1, s22_t2, s22_t3 = 2, 0, 0
s.add(
   ForAll([l22],
      And( Implies(l22 >= 1,
      Or(
         And(l22 >= 1,  mu_p1 + (t1_p1-p1_t1)*s22_t1
         + (t2_p1-p1_t2)*s22_t2 + (t3_p1-p1_t3)*s22_t3
         + l22 * (f_p1 - p1_f) < p1_t3),
         And(l22 >= 1,  mu_p2 + (t1_p2-p2_t1)*s22_t1
         + (t2_p2-p2_t2)*s22_t2 + (t3_p2-p2_t3)*s22_t3
         + l22 * (f_p2 - p2_f) < p2_t3),
         And(l22 >= 1,  mu_p3 + (t1_p3-p3_t1)*s22_t1
         + (t2_p3-p3_t2)*s22_t2 + (t3_p3-p3_t3)*s22_t3
         + l22 * (f_p3 - p3_f) < p3_t3),
)))))

# Sequence 23: f,t1,t2,t1
s23_t1, s23_t2, s23_t3 = 1, 1, 0
s.add(
   ForAll([l23],
      And( Implies(l23 >= 1,
      Or(
         And(l23 >= 1,  mu_p1 + (t1_p1-p1_t1)*s23_t1
         + (t2_p1-p1_t2)*s23_t2 + (t3_p1-p1_t3)*s23_t3
         + l23 * (f_p1 - p1_f) < p1_t1),
         And(l23 >= 1,  mu_p2 + (t1_p2-p2_t1)*s23_t1
         + (t2_p2-p2_t2)*s23_t2 + (t3_p2-p2_t3)*s23_t3
         + l23 * (f_p2 - p2_f) < p2_t1),
         And(l23 >= 1,  mu_p3 + (t1_p3-p3_t1)*s23_t1
         + (t2_p3-p3_t2)*s23_t2 + (t3_p3-p3_t3)*s23_t3
         + l23 * (f_p3 - p3_f) < p3_t1),
)))))

# Sequence 24: f,t1,t2,t2
```

```
s24_t1, s24_t2, s24_t3 = 1, 1, 0
s.add(
   ForAll([l24],
      And( Implies(l24 >= 1,
      Or(
         And(l24 >= 1,  mu_p1 + (t1_p1-p1_t1)*s24_t1
         + (t2_p1-p1_t2)*s24_t2 + (t3_p1-p1_t3)*s24_t3
         + l24 * (f_p1 - p1_f) < p1_t2),
         And(l24 >= 1,  mu_p2 + (t1_p2-p2_t1)*s24_t1
         + (t2_p2-p2_t2)*s24_t2 + (t3_p2-p2_t3)*s24_t3
         + l24 * (f_p2 - p2_f) < p2_t2),
         And(l24 >= 1,  mu_p3 + (t1_p3-p3_t1)*s24_t1
         + (t2_p3-p3_t2)*s24_t2 + (t3_p3-p3_t3)*s24_t3
         + l24 * (f_p3 - p3_f) < p3_t2),
)))))

# Sequence 25: t2,t3,t3
s25_t1, s25_t2, s25_t3 = 0, 1, 1
s.add(
   ForAll([l25],
      And( Implies(l25 >= 0,
      Or(
         And(l25 >= 0,  mu_p1 + (t1_p1-p1_t1)*s25_t1
         + (t2_p1-p1_t2)*s25_t2 + (t3_p1-p1_t3)*s25_t3
         + l25 * (f_p1 - p1_f) < p1_t3),
         And(l25 >= 0,  mu_p2 + (t1_p2-p2_t1)*s25_t1
         + (t2_p2-p2_t2)*s25_t2 + (t3_p2-p2_t3)*s25_t3
         + l25 * (f_p2 - p2_f) < p2_t3),
         And(l25 >= 0,  mu_p3 + (t1_p3-p3_t1)*s25_t1
         + (t2_p3-p3_t2)*s25_t2 + (t3_p3-p3_t3)*s25_t3
         + l25 * (f_p3 - p3_f) < p3_t3),
)))))

# Sequence 26: t2,t3,t2,t2
s26_t1, s26_t2, s26_t3 = 0, 2, 1
s.add(
   ForAll([l26],
      And( Implies(l26 >= 0,
      Or(
         And(l26 >= 0,  mu_p1 + (t1_p1-p1_t1)*s26_t1
         + (t2_p1-p1_t2)*s26_t2 + (t3_p1-p1_t3)*s26_t3
         + l26 * (f_p1 - p1_f) < p1_t2),
         And(l26 >= 0,  mu_p2 + (t1_p2-p2_t1)*s26_t1
         + (t2_p2-p2_t2)*s26_t2 + (t3_p2-p2_t3)*s26_t3
         + l26 * (f_p2 - p2_f) < p2_t2),
```

```
          And(l26 >= 0,  mu_p3 + (t1_p3-p3_t1)*s26_t1
          + (t2_p3-p3_t2)*s26_t2 + (t3_p3-p3_t3)*s26_t3
          + l26 * (f_p3 - p3_f) < p3_t2),
)))))

# Sequence 27: f,t2,t3,t1,t3
s27_t1, s27_t2, s27_t3 = 1, 1, 1
s.add(
   ForAll([l27],
      And( Implies(l27 >= 1,
      Or(
          And(l27 >= 1,  mu_p1 + (t1_p1-p1_t1)*s27_t1
          + (t2_p1-p1_t2)*s27_t2 + (t3_p1-p1_t3)*s27_t3
          + l27 * (f_p1 - p1_f) < p1_t3),
          And(l27 >= 1,  mu_p2 + (t1_p2-p2_t1)*s27_t1
          + (t2_p2-p2_t2)*s27_t2 + (t3_p2-p2_t3)*s27_t3
          + l27 * (f_p2 - p2_f) < p2_t3),
          And(l27 >= 1,  mu_p3 + (t1_p3-p3_t1)*s27_t1
          + (t2_p3-p3_t2)*s27_t2 + (t3_p3-p3_t3)*s27_t3
          + l27 * (f_p3 - p3_f) < p3_t3),
)))))

# Sequence 28: f,t1,t1,t1,t3
s28_t1, s28_t2, s28_t3 = 3, 0, 0
s.add(
   ForAll([l28],
      And( Implies(l28 >= 1,
      Or(
          And(l28 >= 1,  mu_p1 + (t1_p1-p1_t1)*s28_t1
          + (t2_p1-p1_t2)*s28_t2 + (t3_p1-p1_t3)*s28_t3
          + l28 * (f_p1 - p1_f) < p1_t3),
          And(l28 >= 1,  mu_p2 + (t1_p2-p2_t1)*s28_t1
          + (t2_p2-p2_t2)*s28_t2 + (t3_p2-p2_t3)*s28_t3
          + l28 * (f_p2 - p2_f) < p2_t3),
          And(l28 >= 1,  mu_p3 + (t1_p3-p3_t1)*s28_t1
          + (t2_p3-p3_t2)*s28_t2 + (t3_p3-p3_t3)*s28_t3
          + l28 * (f_p3 - p3_f) < p3_t3),
)))))

# Sequence 29: f,t1,t2,t3,t3
s29_t1, s29_t2, s29_t3 = 1, 1, 1
s.add(
   ForAll([l29],
      And( Implies(l29 >= 1,
      Or(
```

```
          And(l29 >= 1,  mu_p1 + (t1_p1-p1_t1)*s29_t1
          + (t2_p1-p1_t2)*s29_t2 + (t3_p1-p1_t3)*s29_t3
          + l29 * (f_p1 - p1_f) < p1_t3),
          And(l29 >= 1,  mu_p2 + (t1_p2-p2_t1)*s29_t1
          + (t2_p2-p2_t2)*s29_t2 + (t3_p2-p2_t3)*s29_t3
          + l29 * (f_p2 - p2_f) < p2_t3),
          And(l29 >= 1,  mu_p3 + (t1_p3-p3_t1)*s29_t1
          + (t2_p3-p3_t2)*s29_t2 + (t3_p3-p3_t3)*s29_t3
          + l29 * (f_p3 - p3_f) < p3_t3),
))))

print(s.check())
print(s.model())
```

Figure A.3: The Generated Fault Identification Problem for Z3.

After running the Python script, the solution generated is presented in Figure A.4.

```
sat
[p2_f = 1, p1_f = 0, f_p3 = 0, p3_f = 0, f_p1 = 1, f_p2 = 0]
```

Figure A.4: The generated solution.

And the graphical representation of the faulty Petri net is depicted in Figure A.5. I have manually inspected the generated faulty Petri net and it can idneed generate the faulty language as expected.

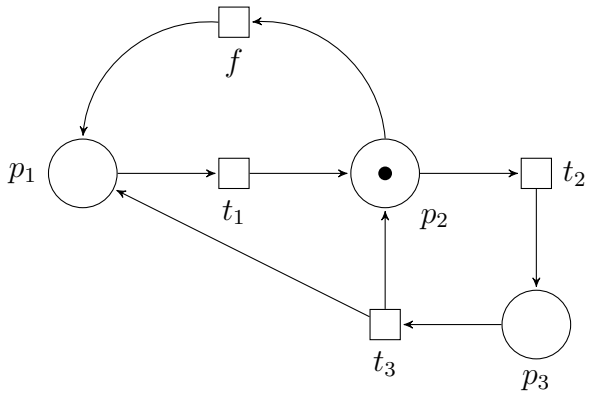

Figure A.5: The faulty net based on the generated solution.

□