ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

DESIGNING, VERIFICATION AND VALIDATION OF RAILWAY SIGNALING SYSTEMS USING COLOURED PETRI NETS

M.Sc. THESIS

Ali ELHAYEK

Department of Control and Automation Engineering

Control and Automation Program

Thesis Advisor: Prof. Dr Mehmet Turan SÖYLEMEZ

DECEMBER 2016

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

DESIGNING, VERIFICATION AND VALIDATION OF RAILWAY SIGNALING SYSTEMS USING COLOURED PETRI NETS

M.Sc. THESIS

Ali ELHAYEK

(504131134)

Department of Control and Automation Engineering

Control and Automation Program

Thesis Advisor: Prof. Dr Mehmet Turan SÖYLEMEZ

DECEMBER 2016

İSTANBUL TEKNİK ÜNİVERSİTESİ ★FEN BİLİMLERİ ENSTİTÜSÜ

DEMİRYOLU SİNYALİZASYON SİSTEMLERİ İÇİN RENKLİ PETRİ AĞLARINI KULLANARAK TASARIM, DOĞRULAMA VE ONAYLAMA

YÜKSEK LISANS TEZI

Ali ELHAYEK

(504131134)

Kontrol ve Otomasyon Mühendisliği Anabilim Dalı

Kontrol ve Otomasyon Mühendisliği Programı

Tez Danışmanı: Prof. Dr Mehmet Turan SÖYLEMEZ

ARALIK 2016

Ali ElHayek, a M.Sc. student of İTU Graduate School of Science Engineering and Technology student ID 504131134, successfully defended the thesis entitled "DESIGNING, VERIFICATION AND VALIDATION USING COLOURED PETRI NETS FOR RAILWAY SIGNALING SYSTEMS", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr Mehmet Turan SÖYLEMEZ** ISTANBUL Technical University	.....
Jury Members :	**Prof.Dr. Salman KURTULAN** ISTANBUL Technical University	.....
	Asst. Prof. Özgür Turay KAYMAKÇI Yıldız Technical University	.....

Data of Submission: 24 November 2016

Date of Defense: 14 December 2016

FOREWORD

I would like to thank my supervisor Prof. Dr Mehmet Turan Söylemez for his excellent guidance and support during this process. I also wish to thank all of the respondents, without whose cooperation I would not have been able to conduct this analysis.

I would like to thanks my friends here in Turkey for their support and for sharing me the time while doing the thesis.

My sister Maha (ablam) deserve a particular note of thanks: your wise counsel and kind words have, as always, served me well. Thanks for my parents for being the light that show me the way.

Last thank will be for my beloved wife Hilal Rabia, for doing everything to help me to reach this point. You are the best thing happened to me in Turkey.

I hope you enjoy your reading.

December 2016

Ali ElHayek

Mechatronics Engineer

TABLE OF CONTENTS

Page

ABBREVIATIONS

APN	: Automation Petri Nets
ASM	: Abstract State Machines
CENELEC	: European Committee for Electrotechnical Standardization
CTC	: Centralized Traffic Control
CPN	: Colored Petri Nets
CTL	: Computational Tree Logic
DES	: Discrete Event System
FSM	: Finite State Machine
IFS	: Infinite Firing Sequences
LTL	: linear time logic
LCC	: Local Command Center
PNs	: Petri Nets
RAMS	: Reliability, Availability, Maintainability and Safety
SIL	: Safety Integrity Level
TCC	: Traffic Command Centre
VDM	: Vienna Development Method

LIST OF TABLES

Page

LIST OF FIGURES

Page

DESIGNING, VERIFICATION AND VALIDATION OF RAILWAY SIGNALING SYSTEMS USING COLOURED PETRI NETS

SUMMARY

Every weekday just in United States of America, more than 7 million people use railways in their transportation. In the same time railway is used widely as reliable freight transportation solution. So in general, railway transportation can be considered as a very vital transportation mean, this importance emerges from the fact that railways are relatively cheap and environmental friendly. This enables them to be the main transportation system in many countries.

Railways systems are exposed to accidents due to huge variety of reasons like signaling system failures, human errors …etc. As railways are used by a huge number of people, the safety of railways became very important issue. This led some governments to interfere by putting standards in order to organize the operations of railway systems.

CENELEC is a safety reference name which states the necessary standards of railway sector and it is composed from the following standards EN 50126, EN 50128 and EN 50129. Based on these standards, Safety Integrity Levels (SILs) were built.

Signalling systems are responsible for the operations of railway systems to ensure the safety of trains and their other components. Signalling ensures optimal control for traffic in order to avoid accidents.

Formal methods have a very important role in software development. Formal methods are method use the discrete mathematic techniques and tools in software and hardware development process, where the mathematical notations are used in the design and the verification of software and hardware systems. The main purpose of using formal methods is to reduce the risky consequences that can occur due to serious specification and design errors by symbolically examine the entire state space of a design Formal methods help in presenting precise record of the created software that's why it is used widely in verification and validation processes

To develop software using formal method, Formal Specifications are used to describe the behavior and properties using formal language and semantics. Formal Language is used to define Rules in a precise manner. Formal language describes the grammar rules and justifies the general algorithms to be used. Semantics provide an accurate mathematical meaning to every statement. These items together will provide a formal model for the system that enables the developers to state the expected properties and then formally verify it

Petri net as the models of DES. Petri net graphs depict structural information about the simple and complex systems. Coloured Petri nets are a high-level Petri nets graphical language. It is based on normal PNs, but colours were added to tokens and places using expressions working with them.

CTL is a branching time tree; the scenarios can be symbolized by hierarchical structure in a graphical form where different scenarios can be applied. CTL* (or ASK-CTL) is used to express the state and the transition properties of the models interviewed by the state space of the coluored Petri net

In this research, a signalling system for a train yard is designed by CPN. The system was verified and validated using model checking which is considered as one of the formal methods. All the processes were performed according to CENELEC to achieve minimum SIL 3.

DEMİRYOLU SİNYALİZASYON SİSTEMLERİ İÇİN RENKLİ PETRİ AĞLARINI KULLANARAK TASARIM, DOĞRULAMA VE ONAYLAMA

ÖZET

Sadece Amerika Birleşik Devletleri'nde hafta içi her gün 7 milyondan daha fazla kişi ulaşımda demiryollarını kullanmaktadır. Aynı zamanda demiryolu, güvenilir yük taşımacılığı yolu olarak yaygın bir şekilde kullanılmaktadır. Genel olarak demiryolu çok önemli bir ulaşım yöntemi olarak düşünülebilir, bu önem ise onun nispeten daha ucuz ve çevre dostu olmasından kaynaklanmaktadır. Bu da birçok ülkede demiryollarının ana ulaşım sistemi olarak kullanılmasına olanak sağlamaktadır.

Demiryolu sistemlerinde esas olarak insan hataları ve sistem arızaları gibi çeşitli sebeplerden dolayı kazalar meydana gelmektedir. Çok sayıda insanın demiryollarını kullanması demiryollarının güvenliğini de bu ölçüde önemli kılmaktadır. Bu durum, bazı hükumetlerin demiryolları sistemlerinin operasyonlarında düzenlemek için standartlar koyarak müdahale etmelerine sebep olmuştur.

Demiryolu sistemlerinde ulaşım ve taşımanın güvenli olarak gerçekleştirilmesini sağlayan en önemli bileşen anklaşman (interlock) sistemidir. Anklaşman sisteminin geliştirilmesinde izlenilecek olan temel adımlar Avrupa Elektroteknik Standardizasyon Komitesi (European Committee for Electrotechnical Standardization - CENELEC) gibi uluslararası komiteler tarafınca hazırlanan güvenlik standartlarında tanımlanmıştır.

EN 50126, EN 50128 ve EN 50129 standartlarından oluşan CENELEC, demiryolu sektöründe gerekli standartları oluşturan güvenlik referansının adıdır.

Geliştirilen sinyalizasyon sisteminin istenilen Güvenlik Bütünlüğü Seviyesi (Safety Integrity Level - SIL) seviyesini sağlayabilmesi için bu güvenlik standartlarınca tavsiye edilen yöntem, teknik ve mimarilerin kullanılması yüksek önem arz etmektedir. Uluslararası güvenlik standartlarının gereksinimlerine ek olarak, sinyalizasyon sisteminin kurulacağı ülkeye ait ihtiyaçlar ve güvenlik kriterleri de göz önünde bulundurulmalıdır.

Yazılım geliştirme süreci başlangıcında yazılımdan beklenen çıktılar veya başka bir deyişle yazılım isterleri oluşturulmalıdır. Sonrasında güvenlik standartlarında tavsiye edilen yöntem ve mimarilerin istenilen SIL seviyesinin sağlanabilmesi için uygun bir şekilde seçilmesi gerekmektedir. Seçilen yöntem ve mimariler yazılım isterlerini eksiksiz sağlayacak şekilde tasarımı gerçekleştirecek olan grup tarafından yazılım geliştirme sürecinde kullanılmalıdır.

Demiryolu sistemlerinde faaliyette bulunan trenler ve demiryolu sistemlerinin diğer bileşenlerinin güvenliğini sağlanması sinyalizasyon sistemlerinin sorumluluğundadır. Sinyalizasyon kazalardan kaçınmak amacıyla, trafik için optimum kontrol sağlamaktadır.

Biçimsel yöntem yazılım geliştirmede çok önemli bir role sahiptir. Biçimsel yöntem ayrık matematik tekniklerinde ve yazılım ve donanım geliştirme sürecinde kullanılan

yöntemlerdir. Matematiksel notasyonlar ise yazılım ve donanım sistemlerinin tasarım ve gerçeklemesinde kullanılmaktadır. Biçimsel yöntemleri kullanmanın temel amacı, bir tasarımın tüm durum uzay modelini sembolik olarak inceleyerek önemli özellikler ve tasarım hataları nedeniyle oluşabilecek riskli sonuçları azaltmaktır. Biçimsel yöntemler oluşturulan yazılımda hassas kayıt sunmaya yardımcı olmaktadır bu nedenle gerçekleme ve doğrulama süreçlerinde yaygın olarak kullanılmaktadır.

Biçimsel yöntem kullanarak yazılım geliştirmek için, biçimsel dil ve semantik kullanılarak davranış ve özellikleri tanımlamak için Kurallı Belirtim kullanılmaktır. Biçimsel Dil kesin bir şekilde Kurallar tanımlamak için kullanılmaktadır. Biçimsel dil, dilbilgisi kurallarını tanımlamaktadır ve genel algoritmaların kullanılmasını haklı çıkarmaktadır. Semantik her ifadeye kesin bir matematiksel anlam sağlamaktadır. Bu öğeler bir araya getirildiğinde ise geliştiricilerin beklenen özellikleri belirtmelerine ve daha sonra biçimsel olarak doğrulamalarına olanak tanıyan sistem için bir biçimsel model sağlayacaktır.

Biçimsel diller ya model tabanlı (Soyut Durum Makineleri, Küme ve sınıf teorisi, otomat tabanlı modelleme ve Gerçek zamanlı sistemler için modelleme dilleri gibi) ya da cebirsel tabanlı olabilmektedir.

Petri ağları bir biçimsel yöntemdir. Petri ağı olayları tanımlanmış kurallara dayalı olarak işleyen ve geçişe izin veren koşulları gösterebilen bir cihazdır.

Petri ağları Ayrık Olaylı Sistemlerin modellerinde kullanılması ve grafiklerin basit ve karmaşık sistemlerin yapısal bilgilerini tasvir etmesi gibi birçok avantaja sahiptir.

Renkli Petri ağları üst düzey Petri ağları grafiksel dildir. Bu normal Petri ağlarına dayanamaktadır ancak jetonlara ve onlarla birlikte çalışan ifadeleri kullanan yerlere renkler eklenmiştir.

Model sınama, sistemin sonlu durum modelini ve biçimsel özelliklerini veren bir otomatik tekniktir. Model sınama bu özelliğin o modeldeki belirli durum için tutulup tutulmadığını sistematik olarak kontrol etmektedir.. Model sınama farklı alalarda kullanılabilen genel bir gerçekleme yaklaşımı olarak düşünülmektedir. Bunun yanı sıra model sınama özellikler için kısmi gerçekleme yapabilmektedir, bu da geliştiricinin önemli olanlara odaklanmasını sağlamaktadır. Model sınama diğer gerçekleme yöntemlerine nazaran daha hızlı olduğu düşünülmektedir.

Dallanan zaman tabanı (Computational Tree Logic) dallanan zaman ağacıdır ki burada senaryolar, farklı senaryoların uygulanabildiği grafik formdaki hiyearşik yapıyla sembolize edilebilmektedir.

Dallanan zaman tabanı* Renkli Petri ağının durum uzayı tarafından gösterilen modellerinin durum ve geçiş özelliklerini ifade etmek için kullanılmaktadır.

İkinci bölümün sonunda Renkli Petri Ağları ve Dallanan Zaman Tabanı* ile ilgili tüm kavramları açıkladıkan sonra, Renkli Petri ağlarının nasıl çalıştığını göstermek ve Renkli Petri Ağlı Dallanan Zaman Tabanının gerçekleme ve doğrulama işleminde kullanılabileceğinden emin olmak için basit tren istasyonuna bir örnek verilmiş ve tasarlanmıştır.

Üstelik sistemin durum uzay modeli gösterilmiştir ve sistemin durum uzay modeli kullanılarak işaretleme konsepti (concept of marking) açıklanmıştır.

Daha karmaşık manevra istasyonu (bu çalışmadaki ana örnek) üçüncü bölümde gösterilmiştir. Manevra istasyonunun bir giriş ve üç çıkışı olmakla birlikte istasyonun hareket kuralları anklaşman tablosuyla tanımlanmış ve bazı kurallar açıklanmıştır. Bu kurallar gerçekleme sürecinde kriter olarak kullanılmıştır.

Manevra istasyonuna ait model Renkli Petri Ağları kullanılarak kurulmuştur ve altı trenin manevra istasyonunda tek yönden geçeceği önerilmiştir. Manevra istasyonunda üç çıkış olduğu için üç rota vardır. Olası tüm senaryolar uygulanmıştır. Tüm senaryoların sonunda tüm trenlerin manevra istasyonunun dışında olması gerekmektedir.

Simülasyon yürütüldükten sonra tüm trenlerin manevra istasyonunun dışında olduğu simülasyonun sonunda gösterilmiştir. Sistemin doğru bir şekilde çalıştığını doğrulamak için sistemin durum uzay modeli ve sıkı bağlı bileşen grafiği oluşturulmuştur ve sonra sistem durum uzay raporu da oluşturulmuştur. Bundan sonra durum uzay modelinin sonuçlarını ve sıkı bağlı bileşen grafiğini test etmek için bazı sorular yazılmıştır.

Sistemi doğrulamak amacıyla üçüncü bölümde belirlenen kurallara göre sorgular da yazılmıştır. Bu sorgular tek tek incelenmiştir ve sistemin belirlenen kurallara uygun olarak çalıştığı gösterilmiştir.

Tasarım, gerçekleme ve doğrulama sürecinde kullanılan bilgisayarın özellikleri 4 GB RAM, i5 CPU 2.53 GHz'dir. Kurulan süreç hafızada yaklaşık 62 MB yer tutmuştur ve CPU'nun %29 unu 32 saniye meşgul etmiştir.

Renkli Petri Ağları ile sistem modeli oluşturmanın kolay olduğu gösterilmiştir. Gerçekleme ve doğrulama süreçleri de kolaydır çünkü hesaplama açısından pahalı değillerdir. Bu yüzden eğer model oluşturulduysa, sistemin tüm işaretlemelerini (marking) taramak uzun zaman ve çaba gerektirmemektedir.

İşaretlemeleri (markings) sınamak sadece Renkli Petri Ağları araçlarını (CPN Tools) kullanarak çok kolay olmamaktadır. Çünkü güncel Petri Ağları araçları (CPN Tools) yeterince geliştirilmemiştir bu yüzden tüm işaretlemelerin(markings) manuel olarak oluşturulması gerekmektedir. Alternatif olarak ise modelin işaretlemelerin oluşturmada karmaşık olan (Graphviz - Graph Visualization Software gibi) bazı araçlar kullanılmaktadır.

Bu yöntemin dezavantajı ise sistemin eksiksizliğini garanti etmemesidir. Sadece belirlenen özellikler sınanmaktadır. Dolayısıyla sistemin tüm önemli özelliklerinin belirlenmesi geliştiricinin sorumluluğundadır.

Dallanan zaman tabanlı Renkli Petri ağları mevcut petri ağı sistemini doğrulamıştır ancak bu sistem için bir PLC kodu oluşturulmamıştır. Bu yüzden doğrudan Renkli petri ağı modeline dayalı PLC kodu oluşturulabilme olasılığının incelenmesi gerekmektedir.

1. INTRODUCTION

Every weekday just in United States of America, more than 7 million people use railways in their transportation [1]. In the same time railway is used widely as reliable freight transportation solution. So in general, railway transportation can be considered as a very vital transportation mean, this importance emerges from the fact that railways are relatively cheap and environmental friendly. This enables them to be the main transportation system in many countries [2].

As any other system, railway systems need to be controlled. In this case controlling system is called signalling system. Signalling system is responsible of operating railway system to ensure the safety of trains, passengers, fright and other components. Signalling system ensures optimal control for traffic in order to avoid accidents [3]. Signalling systems are required to deal with technical failures and ergonomics too [4]. Signalling was started as a very simple process like timetable operation, token signalling. But as the railway sector is developing rapidly, there was a need for the development in signalling consequently. Nowadays very complex technical systems are performing the task of signalling. In modern systems, to have effective signalling system, a number of components have to be synergetic integrated, including:

- Electrical and electronics systems.
- Logical design and software.
- Layout of signals and signal sighting.
- Interactions between signals and drivers.

1.1 Components of a Railway Signalization System

Railway signalization system consists of many components that work together to guarantee the safety of the process (Figure 1.1) .

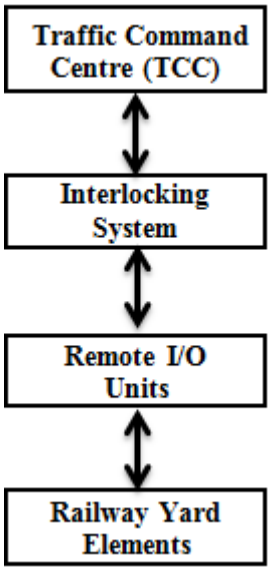


Figure 1.1: Block diagram of signalization system.

1.1.1 Traffic command center (TCC)

Traffic command center is a place where the railway line and train traffic are monitored and controlled. TCC usually is equipped with monitoring system that allows the operator (dispatcher in railways case) to monitor trains and signalling elements along the track [5]. Dispatcher commands, which are mostly route reservation requests, are sent to the interlocking system so they are checked considering the current state of the railway yard.

When routing decisions for a given line are made centrally (TCC), this kind of railway signalling is known as Centralized Traffic Control (CTC) (Figure 1.2.a). When there is a problem in communication, a local command center (LCC) can be used to control the station (Figure 1.2.b).

1.1.2 Interlocking system

When an interlocking system receives a route reservation request or any other request from the dispatcher, it checks the validity of this request by checking the conditions of the related elements on the ground. If the conditions on the ground are compatible with the safety criteria saved in system, acceptance is issued. Otherwise rejection is issued. Figure 1.3 shows the tasks of interlocking system related to different danger types. The safety of the railway system and the trains is the main purpose of interlocking system.

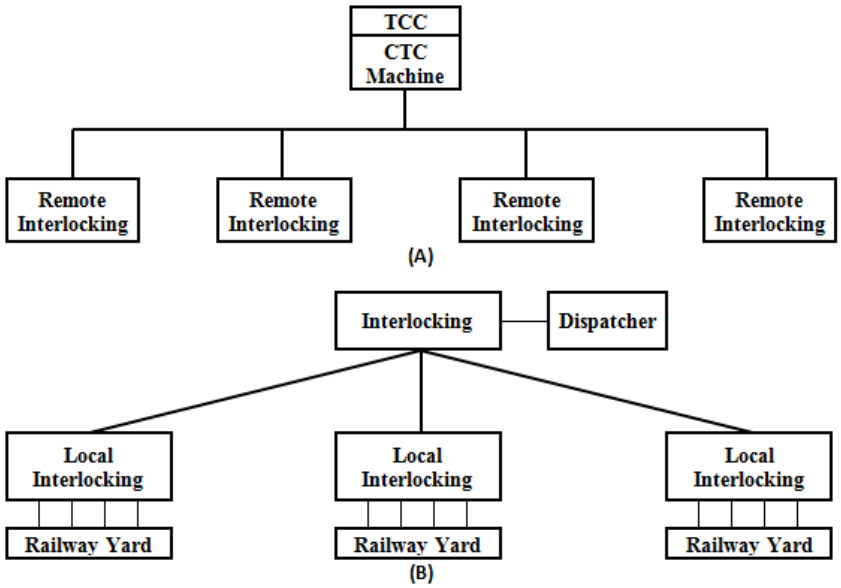


Figure 1.2: Traffic command center.

Danger Type	Collision with				Derailment at	
Danger Reason	External environment	Rail vehicles			Non-continuous guideway locations	continuous guideway locations
Protective Function	Protection against obstacles	Front protection	Following protection	Flank protection	Safety at movable track elements	Speed targeting

Figure 1.3: Dangers and safety measures in railway operations.

1.1.3 Switches

Switches are used in the railways to enable trains to move between tracks. When interlocking system issues an acceptance for a TCC request, switches positions are maintained or changed based on the request. Switches have to be on *normal* or *reverse* position (Figure 1.4).

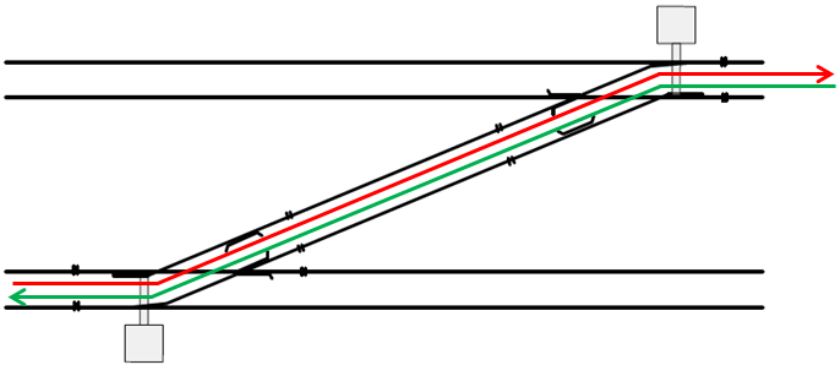


Figure 1.4: Schematic representation of a railway switch [6].

In modern signalization, switch position must be monitored by two sensors, where if position indicators show switch at both positions at the same time, system assumes that the switch is on a faulty state.

1.1.4 Signals

Railway signals (or optical wayside signals) are used along the rail line on certain points to inform train driver about the status of the next railway blocks [3].

Signals are the devices used by interlocking system to authorize the driver to enter a block. So train drivers have to pay attention to signals on the right side (or left- different between countries) with respect to their direction of movement. Despite of the differences between standards of countries in this field, in general red is used to prophet the entrance to a block, green gives complete authority to enter a block, yellow gives authority to enter a block with caution.

In Turkey, additional aspects are used for railways near station areas. The fourth aspect that is clarified in Table 1.1 (a bottom yellow sign) indicates a line change ahead [6]. The dwarf signals are used at the exits of secondary lines of the railway fields. They indicate that the train will be changing lines through a switch (or a set of switches).

Table 1.1: The types and the definitions of common signals [6].

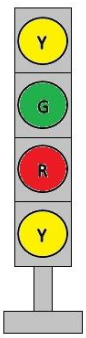
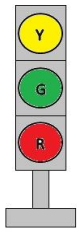
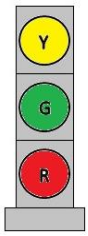
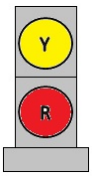
Type of Signal	Schematic	Color	Definitions
Four-aspect Tall Signal		Green (G)	The up coming two blocks are available; the train can keep on.
		Yellow (Y)	The up coming block is available, but the second block is not. Train can Keep but carefully.
		Red (R)	Stop, the coming block is not available.
		Yellow-Green(YG)	A diverge ahead and the coming two blocks are available.
		Yellow-Yellow (YY)	A diverge ahead and the coming block is available. But the second block is not.
		Yellow-Red (YR)	Keep on very carefully (stop when necessary).

Table 1.1 (Continued): The types and the definitions of common signals [6].

Three-aspect Tall Signal		Green (G)	Coming two blocks are available; the train can keep on.
		Yellow (Y)	The coming block is available, but the second block is not. Train can Keep on but carefully.
		Red (R)	Stop, the coming block is not available.
Three-aspect Dwarf Signal		Green (G)	Coming two blocks are available; the train can keep on.
		Yellow (Y)	The coming block is available, but the second block is not. Train can Keep but carefully.
		Red (R)	Stop, the coming block is not available.
		Yellow-Red (YR)	Keep on very carefully (stop when necessary).
Two-aspect dwarf Signal		Yellow (Y)	The coming block is available, but the second block is not. Train can Keep but carefully.
		Red (R)	Stop, the coming block is not available.
		Yellow-Red (YR)	Keep on very carefully (stop when necessary).

1.1.5 Track circuits and axle counters

Track circuits and axle counters are used to detect the location of the train on the rail. Different techniques and combinations are used to detect the train location. More information about detection technics are explained in [3].

As signalling systems are composed from multi integrated systems, the process of proving system safety became complex [7, 8]. Moreover railways systems are exposed to accidents due to huge variety of reasons like signaling system failures, human errors …etc. As railways are used by a huge number of people, the safety of railways became very important issue. This led some governments to interfere by putting standards in order to organize the operations of railway systems [1] [2].

1.2 Railway Standards

Developing hardware section to be compatible with logical design and software of interlocking system is one from the hardest tasks in signalling system design

[9]. CENELEC- (European Committee for Electrotechnical Standardization) - is a safety reference name which states the necessary standards of railway sector and it is composed from the following standards EN 50126, EN 50128 and EN 50129. Based on these standards, Safety Integrity Levels (SILs) were built [10]. SIL indicates the maximum failure rate that can be accepted [7]. Table 1.2 shows failure rates for different SILs. For a system to be SIL 4, it means that during 10,000 working years the system must not face any failure, where λ in Table 1.2 indicates to failure rate.

Table 1.2: Failure rates for different SILs [10].

SIL	Failure Rate
4	$10^{-9}/h \leq \lambda < 10^{-8}/h$
3	$10^{-8}/h \leq \lambda < 10^{-7}/h$
2	$10^{-7}/h \leq \lambda < 10^{-6}/h$
1	$10^{-6}/h \leq \lambda < 10^{-5}/h$

In general standards are divided to:

- Basic standard:
    1. IEC 61508: Functional safety of electrical/ electronic/programmable electronic safety-related systems [11].
- Specific CENELEC standards derived from IEC 61508:
    1. EN 50126-1:2012 - Railway applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS) [12].
    2. EN 50129:2003 - Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling [13].
    3. EN 50159:2010 - Railway applications - Communication, signalling and processing systems - Safety-related communication in transmission systems [14].

4. EN 50128:2011 - Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems.

Software development processes can be done according to these standards (Figure 1.5) [15].

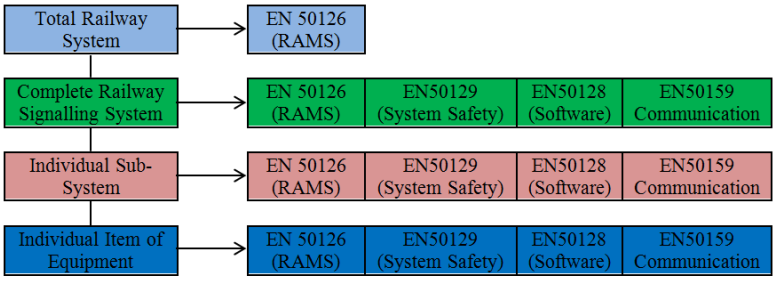


Figure 1.5: Scope of th main CENELEC railway application standards.

Conventional V-model was created for software development purposes. V-Model illustrates how to build the software of a signalling system using EN 50128 (Figure 1.6).

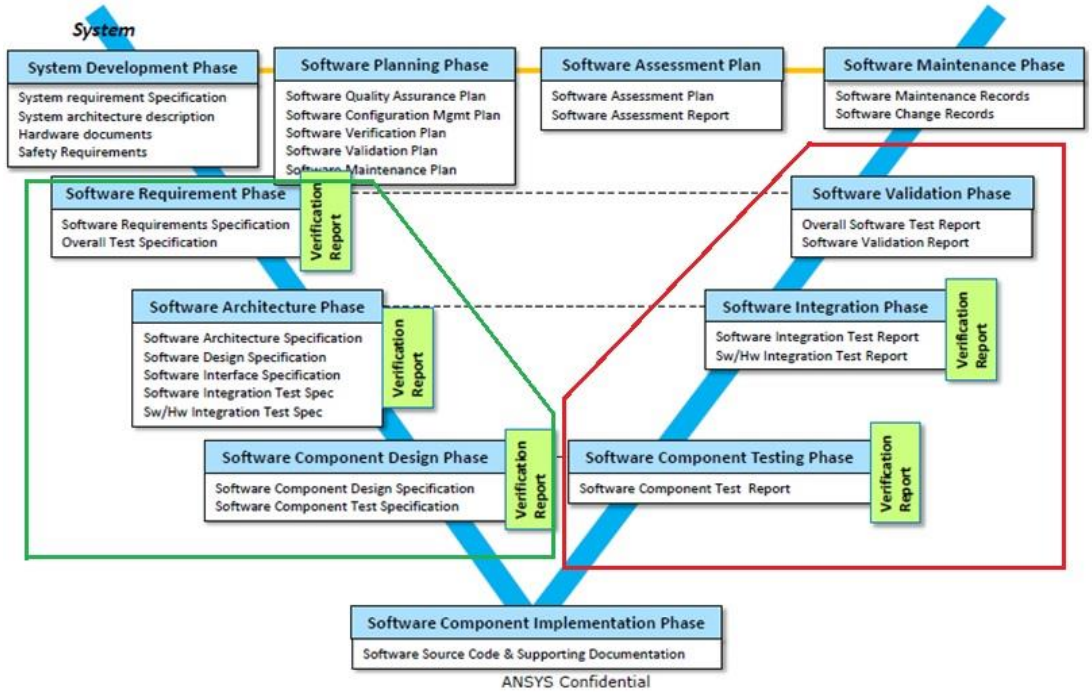


Figure 1.6: V-Model life cycle.

The model can be divided into left section and right section. The first section (surrounded by green) of the model shows that must go under verification to make sure that it is functioning in a proper way, second section of V-Model

(surrounded by red) shows that system must be validated too in order to see if it performs the right function or not. If any negative result in validation appears at any point, the developer is required to repeat the developing steps from the point related point on the first section of the model (as it can be seen in Figure 1.6 every point in validation section is related to a point in verification process). EN 50128 defines the proper criteria for every node in V-Model.

Table 1.3 shows some details of these producers. Following these procedures will help to achieve software which is analyzable, testable, verifiable and maintainable [15]. As it is noticed, formal methods have a very important role in software development, so in the next chapter Formal methods will be reviewed.

Table 1.3: Some nodes of V-Model.

Node Name	Method
Software Requirements Specification	Formal Methods
	Modelling
	Structured methodology
	Decision Tables
Software Architecture	Defensive Programming
	Fault Detection & Diagnosis
	Error Detecting Codes
	Failure Assertion Programming
	Diverse Programming
	Memorizing Executed Cases
	Software Error Effect Analysis
	Graceful Degradation
	Information Encapsulation
	Fully Defined Interface
	Formal Methods
	Modelling
	Structured Methodology
	Modelling supported by computer aided design and specification tools

Table 1.3 (Continued): Some nodes of V-Model.

Software Design and Implementation	Formal Methods
	Modelling
	Structured methodology
	Components
	Analyzable Programs
	Strongly Typed Programming Language
	Structured Programming
	Programming Language
	Language Subset
	Procedural programming

2. FORMAL METHODS

Formal methods are method use the discrete mathematic techniques and tools in software and hardware development process, where the mathematical notations are used in the design and the verification of software and hardware systems. Formal methods help in presenting precise record of the created software [19]. Formal methods are well-formed statements in a mathematical logic in which the formal verifications are rigorous deductions. The main purpose of using formal methods is to reduce the risky consequences that can occur due to serious specification and design errors by symbolically examine the entire state space of a design [20, 21].

To develop software using formal method, **Formal Specifications are** used to describe the behavior and properties using formal language and semantics. **Formal Language** is used to define Rules in a precise manner. Formal language describes the grammar rules and justifies the general algorithms to be used. **Semantics** provide an accurate mathematical meaning to every statement. These items together will provide a formal model for the system that enables the developers to state the expected properties and then formally verify it [22, 23].

Formal methods can be classified based on their main purpose; it can be descriptive or analytic. **Descriptive methods** focus on specifications as a tool for review and discussion, while **analytic methods** focus on the utility of specifications as a mathematical model for analyzing and predicting the behavior of (hardware and software) systems. Analytic formal methods work on emphasizing mechanization and general design specification languages capable of supporting efficient automated deduction [24]. Formal language can be based on mathematical model or on a standardized programming or specification language. Formal specification can be (partly) executable. Generally only subsets of formal specification languages, e.g. of Z and VDM, are machine executable [25]. Formal languages can be either model-based (such as abstract state machines (ASM), Set and category theory, automata-based modelling and modelling languages for Real-time systems) or algebraic based, for more information reader is suggested to refer to [19].

2.1 Discrete Event System

To study the discrete event system (DES), appropriate models must be developed, that accurately defines the behavior of these systems and provides a framework to help in satisfying the design targets, controlling the system, and evaluating the performance of the system. Automaton was conducted for these purposes. As aforementioned, automata and its related languages can be used to in formal way to study the logical behavior of DES.

2.1.1 Automata

An automaton can be defined as a device that is capable of representing a language according to well defined rules.

A **language** defined over an **event set *E*** is a set of finite-length strings formed from events in *E*.

- As an example, let $E = \{a, b, c\}$ be the set of events. Then languages may be defined as:

$L1 = \{\varepsilon, a, acb\}$ that consisting of three strings only. (ε denotes empty string).

$L2 = \{$all possible strings of length 3 starting with event $b\}$ which contains nine strings.

$L3 = \{$all possible strings of finite length which start with event *c*.

An automaton consists from places (which represent states) and transitions (which represent events). To understand how automata works, Let the event set be E = {a, b, c}. Consider the state transition diagram in Figure 2.1, where nodes represent **states** (places) $X = \{x_1, x_2, x_3\}$, and labeled **arcs** represent **transitions** between these states. This directed graph provides a description of the dynamics of an automaton [20].

To describe how the transitions between states occur, a function must be defined. For the case in Figure 2.1 function can be defined as follows:

$$f: X \times E \rightarrow X$$

This means, when a state is associated with an appropriate event it will led to a state.

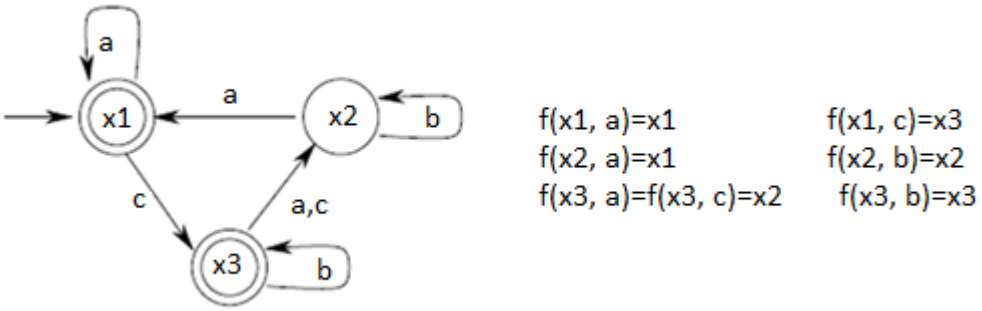


Figure 2.1: State transition diagram.

For the state transition diagram the notation $f(x_2, a) = x_1$ means that if the automaton is in state x_2, then upon the "**occurrence**" of event "a", the automaton will make an instantaneous **transition** to state x_1. More than one event can be written in the same notation, as example

$f(x_1, cba) = f(f(x_1, cb), a) = f(f(f(x_1, c), b), a) = f(f(x_3, b), a) = f(x_3, a) = x_2$.

Automaton can be defined formally as in [20]. In this work another method for DES modelling will be used, it is Coloured Petri Nets.

2.1.2 Petri nets (PNs)

It was firstly introduced by Carl Adam Petri in 1960`s, somehow petri nets is related to automata as they represent the transition function of DES [20]. PNs have some advantages over automata in both graphical and mathematical features- automaton can always be represented as a Petri net; on the other hand, not all Petri nets can be represented as finite-state automata [9]. A Petri net is a device that operates events based on defined rules and can show the conditions that enable a transition. PNs have a lot of advantages that motivate considering Petri net as the models of DES. Petri net graphs depict structural information about the simple and complex systems [20]. In general, PNs consist of three types of components: places (circles), transitions (rectangles) and arcs (arrows). Where places represent possible states of the system, transitions are events or actions which cause the change of state, and every arc simply connects a place with a transition or a transition with a place. Next section shows the formal definition of petri nets.

2.1.2.1 Formal definition of petri nets

Petri nets are defined in the literature by [26];

$$PN = P, T, A, W, M_0 \quad (2.1)$$

- P: {P_1, P_2 … P_n}, finite set of places,
- T: {t_1, t_2 … t_m}, finite set of transitions,
- A ⊆ (P x T) ∪ (T x P) is a set of arcs,
- W: A → {1 2 3 …} is a weight function,
- M_0: P → {0 1 2 3 …} is the initial marking
- P ∩ T = Ø and P ∪ T ≠ Ø.

2.1.2.2 Basic properties of petri nets

- A transition t is said to be enabled if each input place P of t is marked at least W(P, t) tokens, where W(P, t) is the weight of arc from place P to transition t.

$$x(P_i) \geq W(P_i, t_j) \; for \; all \; P_i \in I(t_j) \quad (2.2)$$

Where $x(P_i)$ is the number of tokens on i^{th} place and $I(t_j)$ is the sets of input places of transition t_j. T_0 in Figure 2.2.a is enabled as the number of tokens in P_0 and P_1 is more than the weight of the related arcs, but for T_0 in Figure 2.2.b is not enabled as the number of tokens in P_0 and P_1 is less than the weight of the related arcs.

- An enabled transition may or may not fire (depending on whether or not the event actually takes place).

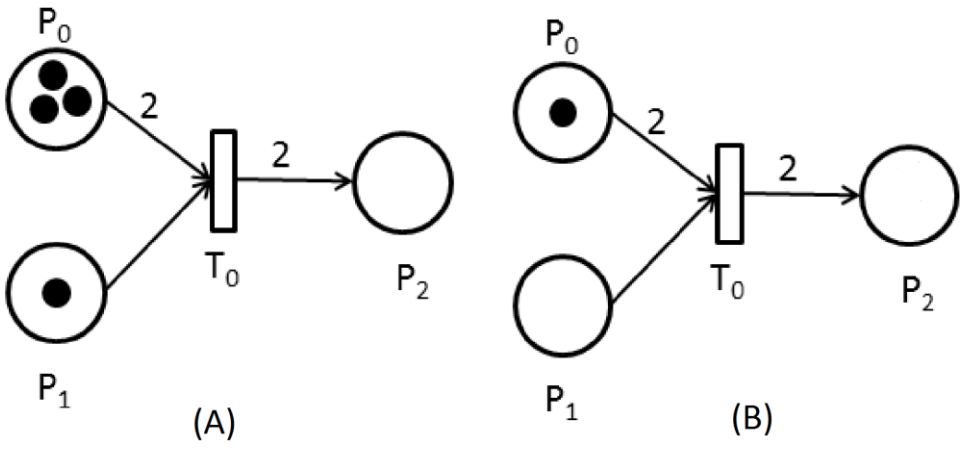


Figure 2.2: Weight of arcs.

- A firing of an enabled transition t removes W(P, t) tokens from each input place P of t and adds W(t, P) tokens to each output place P of t, where W(t, P) is the weight of the arc from t to P.

- where $x'(P_i)$ is the number of tokens on i[th] place after the firing of transition j [26]. If the inequality in Equ.2 is applied for P0 in Figure 2.3, $x'(P_0)$=1, left part of inequality is (3-2-0=1).
- Other models were built based on petri nets like Automation Petri Nets (APN) [21], other models were classified as high level petri nets like Stochastic Petri Nets, Colored Petri Nets and Object Oriented Petri Nets [27]. In this work Coloured Petri Nets will be used.

$$x'(P_i) \geq x(P_i) - W(P_i, t_j) + W(t_j, P_i) \quad (2.3)$$

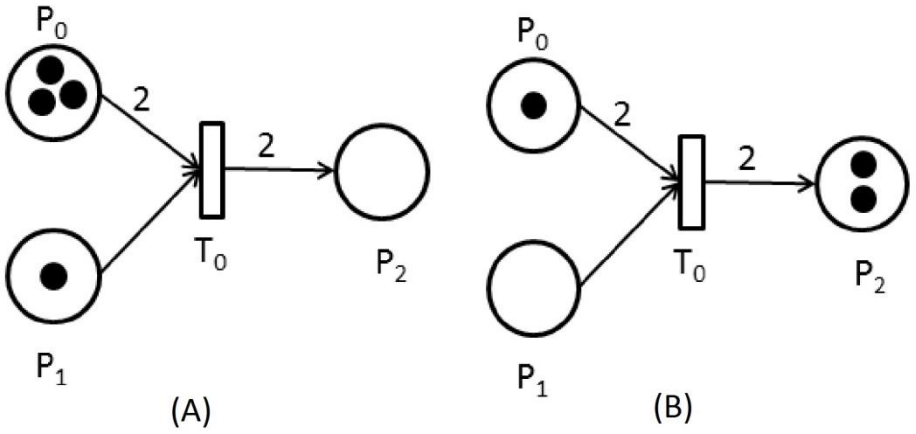


Figure 2.3: Number of tokens befor and after firing.

2.1.3 Coloured petri nets (CPNs)

Coloured Petri nets are a high-level Petri nets graphical language. It is based on normal PNs, but colours were added to tokens and places using expressions working with them. They have well-defined semantics and well known for specifying distributed and concurrent systems which makes it very efficient formal modelling technique [28-31]. As CPN tokens have colours (data value), this helps in reducing the number of places if it is compared with normal petri net [32].

CPN can be classified as non-hierarchical Colored Petri Net and hierarchical Colored Petri Net.

2.1.3.1 A non-hierarchical colored petri net

It is a nine-tuple

$$CPN = (P, T, A, \Sigma, V, C, G, E, I) \quad (2.4)$$

where [28, 29]:

1. P is a finite set of places.

2. T is a finite set of transitions such that P ∩ T = ∅,

3. A ⊆ P × T ∪ T × P is a set of directed arcs.

4. Σ is a finite set of non-empty color sets.

5. V is a finite set of typed variables such that Type[v] ∈ Σ for all variables v ∈ V.

6. C : P → Σ is a color set function that assigns a color set to each place.

7. G : T → EXPRV is a guard function that assigns a guard to each transition t such that Type[G(t)] = Bool.

8. E : A → EXPRV is an arc expression function that assigns an arc expression to each arc such that Type[E(a)] = $C(p)_{MS}$[1], where p is the place connected to the arc a.

9. I : P → EXPR∅ is an initialization function that assigns an initialization expression to each place p such that Type[I (p)] = $C(p)_{MS}$.

2.1.3.2 Hierarchical colored petri net

It is a four-tuple

$$\mathrm{CPN_H} = (\mathrm{S}, \mathrm{SM}, \mathrm{PS}, \mathrm{FS}) \quad (2.5)$$

Where [29]:

1. *S* is a finite set of *modules*. Each module is a *Colored Petri Net Module*

$$s = \left((P^s, T^s, A^s, \Sigma^s, V^s, V^s, C^s, G^s, E^s, I^s) T^s_{sub}, P^s_{port}, PT^s\right)$$

It is required that $(P^{s1} \cup T^{s1}) \cap (P^{s2} \cup T^{s2}) = \emptyset$ for all $s1, s2 \in S$ such that $s1 \neq s2$.

2. $SM : T_{sub} \rightarrow S$ is a submodule function that assigns a *submodule* to each substitution transition. It is required that the *module hierarchy* is acyclic.
3. *PS* is a port-socket relation function that assigns a *port-socket relation* $PS(t) \subseteq P_{sock}(t) \times P^{SM(t)}_{port}$ to each substitution transition *t*. It is required

[1] *MS* refers to "multiset."

that $ST(p) = PT(p'), C(p) = C(p')$ and $I(p) <>$ for all $(p, p') \in PS(t)$ and all $t \in T_{sub}$.

4. $FS \subseteq 2^P$ is set of non-empty *fusion sets* such that $C(p) = C(p')$ and $I(p) <>= I(p') <>$ for all $p, p' \in fs$ and all $fs \in FS$.

Appendix A explains how CPN works. To see more explanations about hierarchical and non- hierarchical colored petri net, reader is suggested to refer to [30]. Colored petri net has mathematical structure and mathematical properties.

2.1.3.3 CPN properties

A. Home Properties
   1. Home Marking

      A marking $m \in N^m$ of a Petri net is a home-marking if it is reachable from all reachable markings. A set of markings $M \subseteq N^m$ of a Petri net is a home space if for all reachable marking m a marking in M is reachable from m [25].

B. Liveness Properties
   1. Dead Marking: when CPN reaches a marking that does not lead to other marking\s, this marking is known as dead marking [30, 31].
      Given a marked net <N, m_0> let m ∈ R(N,m_0) be a reachable marking. It is said that m is a dead marking if no transition is enabled at m. i.e., if <N, m> is dead. A marked net <N, m_0> is deadlocking if there exists a dead reachable marking [33]. If there are more than one dead marking it means that the system has no home marking.
   2. Dead Transition Instances: When a transition is not enabled within any marking, this transition is considered as dead transition [30].
   3. Live Transition Instances: if there is exist an infinite sequence of markings where a transition can happen infinitely and can be reached from any marking and can be reached from the initial marking, there is a live transition in the system [30].

C. Fairness Properties

   Fairness is only relevant if there are infinite firing sequences (IFS) [34]. And they provide information about how often a transition can occur [32].
   1. Impartial Transition Instances: It is the set of transitions that form a main part in the infinite sequences, and by remove or deactivate this transition the

infinite sequences are over [30]. As it can be seen from Figure 2.4 that all the transitions are occur in every IFS [34]. All of the Transitions in Figure 2.4 are considered as impartial transition.

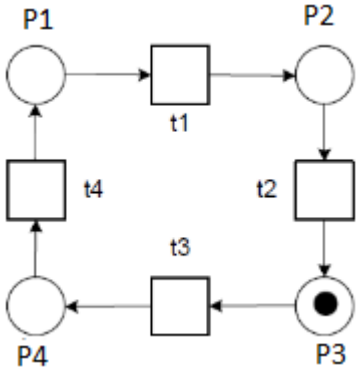


Figure 2.4: Impartial transition.

2. Fair Transition Instances: the transition occurs infinitely often in all infinite occurrence sequences where it is infinitely often enabled [32]. t1 in Figure 2.5 is fair transition instance.
3. Just Transition Instances: when the transition occurs infinitely often in every IFS where t is continuously enabled from some point onward [34]. t6 in Figure 2.5 is just transition instance.
4. Transition Instances with No Fairness: not just, i.e., there is an IFS where t is continuously enabled from some points onward and does not fire anymore [34]. t2, t3, t4 and t5 in Figure 2.5 have no fair.

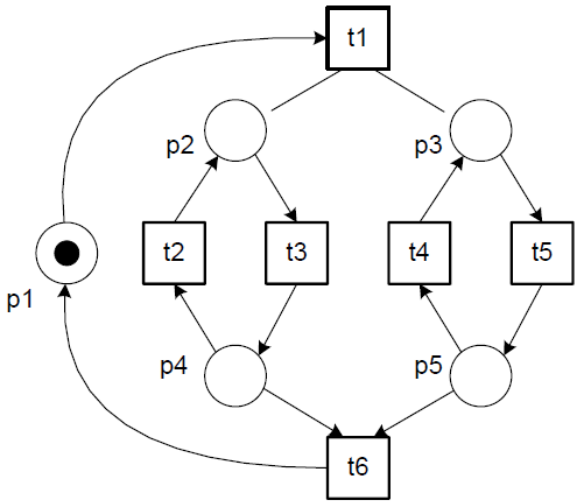


Figure 2.5: Fairness properties.

D. Boundedness: It refers to the property of token on a place not exceeding a given positive integer.

Place $p_i \in P$ in Petri net N with initial state x_0 is said to be k-bounded, or k-safe, if $x(p_i) \leq k$ for all states $x \in R(N)$, that is, for all reachable states [20].

2.1.3.4 CPN and system behavior

State space analysis is used to investigate the functional behavior of systems. System events sequences can be inspected and visualized as a state space. Nodes in the state space represent states, and arcs represent state changes. Every node represents a state of the system (the values of the places can be viewed), and every state of the system is defined as marking (a marking represents a picture of the nodes of the state space). Transfer between markings requires some proper event to occur [31, 35]. CPN provides strongly connected component (SCC) graph which shows a set of nodes that are strongly connected to each other's [30]. One useful method in formal verification is to generate all possible space state of the system and examine them. CPN is supported by ML code to help in analyzing space states; moreover it is augmented by ASK-CTL as external tool for the same purpose, this will be explained later.

2.2 Model Checking

Model checking is an automated technique that gives a finite-state model of a system and a formal property. it systematically checks whether this property holds for a given state in that model. Model checking is considered as a general verification approach that can be used in different fields. Moreover, it is capable of doing partial verification for properties, which enables the developer to focus on the important ones. Model checking is considered fast relative to other verifications methods. But in the same time it just checks the stated requirements, which means that it can just validate the stated criteria without guarantee of completeness. One of its disadvantages, it suffers from state-space explosion (the number of state-space exceed the ability of computer) [36], but for applications such as the interlocking system if state-space explosion is exists it means that there is a problem, that`s why it looks as very suitable solution for interlocking system applications.

2.2.1 Model checking procedures

The process of model checking was initiated by [37]:

1. Modelling: Constructing a model that is accepted by a model checking tool (CPN will be used to build the model in this research).
2. Specifications: The properties under concern that are needed to be stated. Usually the specifications are given in some logical formalism like **temporal logic**. It is the developer responsibility to state all important properties to be validated. Model checker will validate the stated properties if it is valid even if the system in general is not valid (completeness is not guaranteed).
3. Verification: In general the verification is completely automatic. But sometimes the developer is required to be involved into the process to analyze the results. Developers can be supported with an error trace in case of negative results.

2.2.2 Temporal logic

Temporal logic is a formalism for describing sequences of transitions between states in a reactive system (system react to certain events). There are several types of temporal logic. Linear Time Logic (LTL) is a special case - infinite sequence of states where each point in time has a unique successor [23, 37]. But LTL does not permit quantifying along paths, e.g. state the existence of a path satisfying a specific property. Computational Tree Logic (CTL) is an extension of LTL which permits quantifying along paths by using universal and existential quantifiers to the modal operators.

2.2.3 Computational tree logic

CTL is a branching time tree; the scenarios can be symbolized by hierarchical structure in a graphical form where different scenarios can be applied [38, 39]. The name comes from considering paths in the computational tree gotten by unwinding the FSM. Due to its ability in showing possesses safety or liveness properties it is used in formal verification. For example CTL is used to realize home properties, liveness properties and fairness properties. CTL* is high level CTL (it can quantify paths and deal with temporal operators).

Traditionally, temporal logics use Kripeke structure model concurrent systems, [36, 37] show formal definition of Kripeke structure and its properties. The tree of CTL* is formed by unwound the Kripeke structure into an infinite tree rooted at the initial state. Figure **2.6** shows an example of unwinding a graph into a tree. Paths in this tree

represent all possible computations starting from the initial state of the modeled system. Then the temporal operators describe the path properties through the tree.

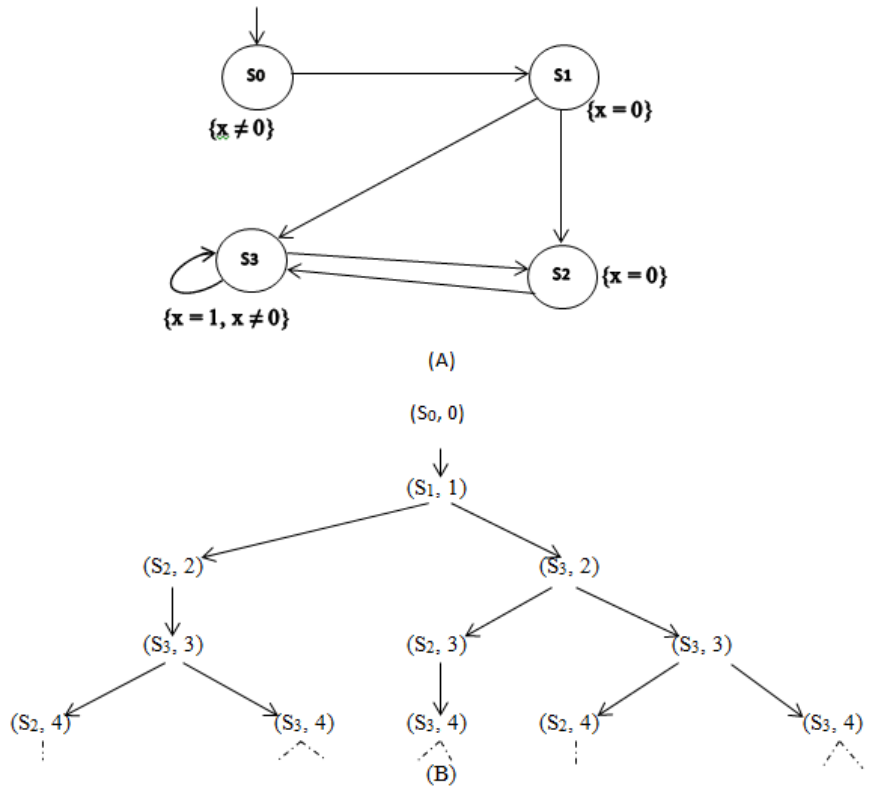


Figure 2.6: Unwind state graph to obtain infinite tree.

2.3 CPN and CTL*

CTL* (or ASK-CTL) is used to express the state and the transition properties of the models interviewed by the state space of the coluored Petri net [26]. It can be applied over the state space of CPN and can deal with state information and transition information. Other tools like model checker were added which can help in checking the formula against the current state space, and returns the true value of the given formula [40].

2.4 Demonstration for Railway Application

In Appendix B, a model for train yard was built by CPN tools, the system was verified then some criteria were proposed to be validated.

3. CASE STUDY

A train layout will be studied in this Chapter. Operations rules and restrictions will be discussed as well.

3.1 Train Yard Details

The layout contains 3 lines, it consists of 5 sections. Layout is protected by signal B2D (Figure **3.1**).

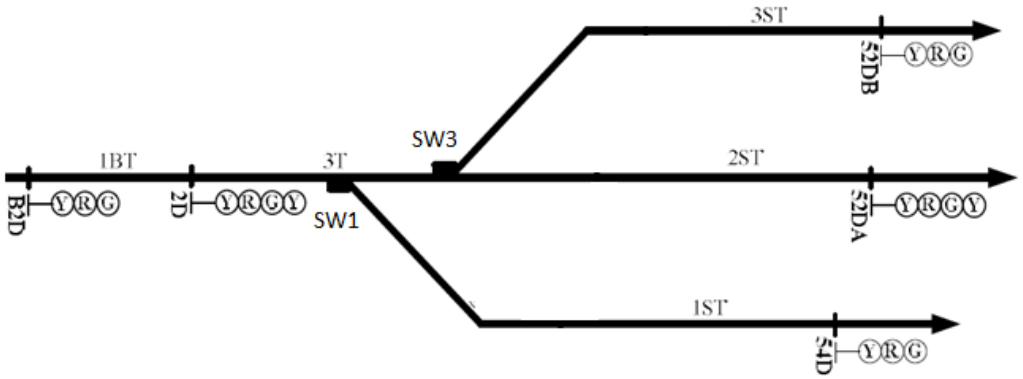


Figure 3.1: Railway yard.

Operation after train entering section 1BT is going to be discussed. Table 3.1 shows the rules for movements in the yard. One direction flow will be under consideration (from left to right). Authorization by signal 2D is based on next signal related to the reserved rout (in 1BT-2ST 2D authorization type is based on signal 52DA).

3.2 Verification and Validation Criteria

After the system is built, the system must be verified to check if it is working in a proper way. It must not contain non-safe deadlock and live-lock. Then the system is going to be validated according to next rules.

1. No more than one permission can be given in 1BT at the same time.
2. B2D is not giving permission if a rout is reserved or there is another train in 1BT.

Table 3.1: Interlocking table.

Rout No.	Route Selection	Position of Switches	2D	Next Signal Light
1	1BT-1ST	SW1 reverse	YY	G
				Y
			YG	R
2	1BT-2ST	SW1, SW3 normal	G	G
				Y
			Y	YG
				YY
				R
3	1BT-3ST	SW1 normal SW3 reverse	YY	G
				Y
			YG	R

3. No train leaves 1BT without a reserved rout.
4. For 1BT_1ST to be reserved, none of the routs must be reversed, a train must be on section 1BT, section 1ST must be free, section 3T must be free and request to reserve 1BT_1ST from TCC must be delivered.
5. For 1BT_2ST to be reserved, none of the routs must be reversed, a train must be on section 1BT, section 2ST must be free, section 3T must be free and request to reserve 1BT_2ST from TCC must be delivered.
6. For 1BT_3ST to be reserved, none of the routs must be reversed, a train must be on section 1BT, section 3ST must be free, section 3T must be free and request to reserve 1BT_3ST from TCC must be delivered.
7. No train can enter 3T without permission from 2D.
8. In 1BT_2ST reservation case, for 2D to give permission, SW1 and SW3 must be normal, 1BT_2ST rout must be reserved and 52DA is not red.
9. In 1BT_1ST reservation case, for 2D to give permission, SW1 must be reversed, 1BT_1ST rout must be reserved and 54D is not red.
10. In 1BT_3ST reservation case, for 2D to give permission, SW1 must be normal, SW3 must be reversed, 1BT_3ST rout must be reserved and 52DB is not red.
11. No train can enter 3T if there is another train in the section.
    (Next rules will be applied for 1ST, 2ST and 3ST).

12. While train is going out from 3T section, it must be sure that the train is going to the appropriate section
13. For 52DA to give permission 1BT_2ST rout must be reserved.

4. IMPLEMENTATION

This chapter will review the model details and discuss the verification and validation procedures.

4.1 Model Building

4.1.1 Main layout

The system was divided into modules in order to ease the modeling and bugging of a system. These modules were connected to each other's in a layout (Figure 4.2**)**. The modules are shown in Table 4.1.

Table 4.1: Modules names.

Module Name	Name Meaning
S1BT	Section 1BT
S3T	Section 3T
S1ST	Section 1ST
S2ST	Section 2ST
S3ST	Section 3ST
SW1	Switch 1
SW2	Switch 3
Interlocking	Interlocking processes

Declarations in Figure 4.1 were used to define the variables related to each color. The colors shown in Figure 4.2 have no operational meaning; they were just used to illustrate the model. Places in layout apart from "Depo", "Out1ST", "Out2ST" and "Out3ST" are used to connect between the modules. When simulation is started, all tokens of trains are outside 1BT (at place "Depo"), and when it is finished tokens of trains will be on "Out1ST", "Out2ST" and "Out3ST" places. If simulation is completed and at least one train token is inside a module, this means that the model

has a fault. The module was hierarchically built (2.1.3.2 Hierarchical colored petri net). From now and on “token” are going to be used insisted “train token”.

```
colset BOOL = bool;
var b:BOOL;
var b2:BOOL;
var b3:BOOL;
colset INT = int;
var n:INT;
colset INTINF = intinf;
colset STRING = string;
colset TCC=index rout with 1..3;
var tcc:TCC;
colset INTxTCC = product INT * TCC;
colset TRAIN=index tr with 1..5;
var tt:TRAIN;
colset INTxTRAIN = product INT * TRAIN;
colset SIG=with r|g|y|yy|yg|yr;
var sign:SIG;
var sign2:SIG;
colset SW=with  actv;
var sw:SW;
```

Figure 4.1: Declarations.

4.1.2 Section 1BT

When a train leaves “Depo”place, it enters the first section “1BT”. Layout is connected with S1BT through “Depo” place**.** Figure 4.3 shows the model of 1BT. The places surrounded by red are connected with interlocking module. For analysis purpose, trains were numbered and transition “App1BT” was connected to interlocking module.

When a train arrives to place “P02_App1BT”- assuming the train driver is obeying the rules - it waits there till permission is issued from signal B2D which is represented by place “B2D”.

Before B2D gives authority to enter section 1BT, it must be checked that there no reserved route and that the section is empty. B2D becomes red again when the train is inside the section. Figure 4.4 shows how these procedures are performed.

It must be noticed that when a place has a socket “In” it means that this place is controlled from another module, vice versa when it has socket “Out” it means that it is controlled in current module and its values can be read from another module/s. While the train is passing transition “IntoS1BT”, the value of place “At1BT_Status” is replaced from “false” by “true”. At this stage the train cannot leave 1BT as long as there is no permission given from signal 2D. But before giving authorization, a route must be reserved.

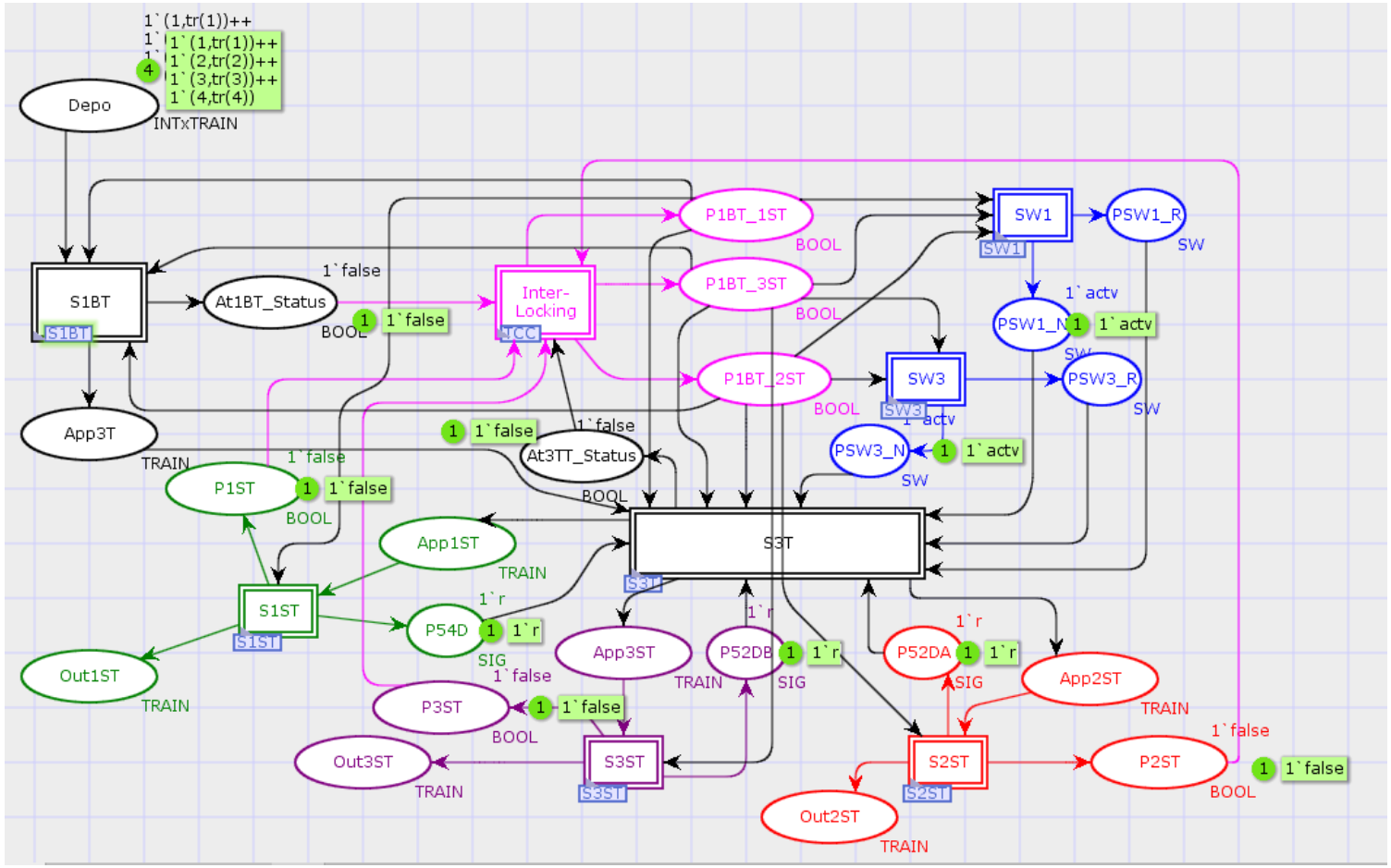


Figure 4.2: Model layout.

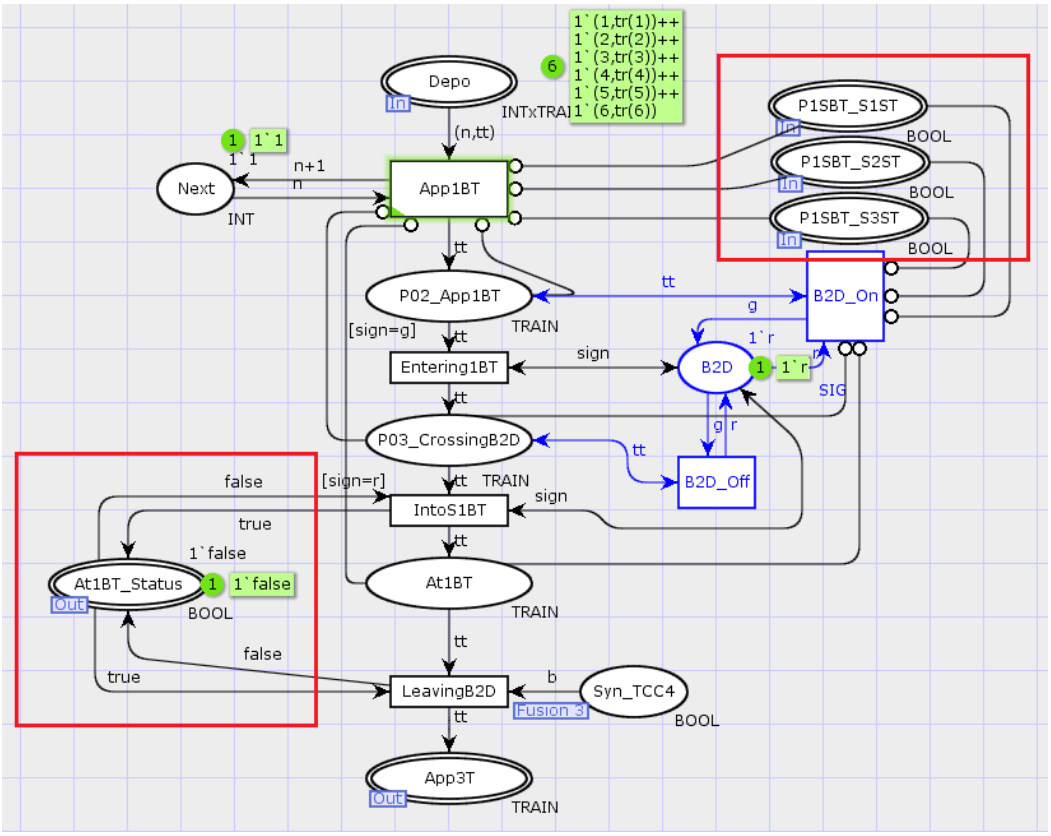


Figure 4.3: 1BT section model.

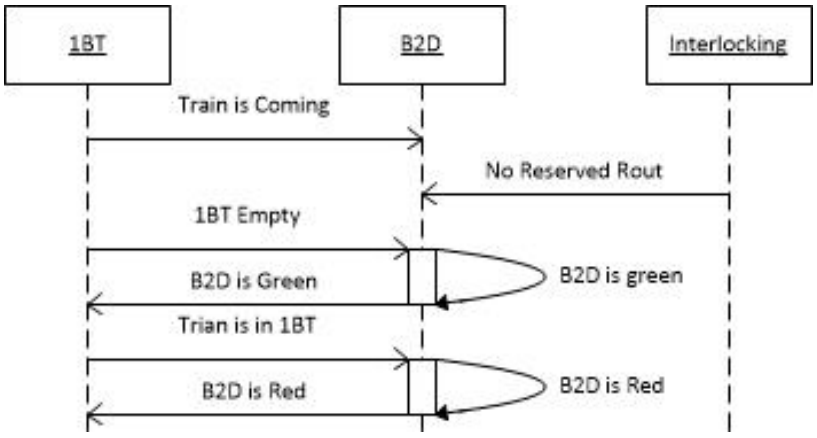


Figure 4.4: B2D changing color sequential diagram.

4.1.3 Interlocking system

There are three possibilities for routes as it was shown in Table 3.1. 1BT-2ST will be considered from now and on. Interlocking module is shown in Figure 4.5**.** Elements surrounded by red represent TCC elements, from where the route reservation requests will be sent. Transition “TCC” will not be enabled if there is a reserved route or there is no train in section 1BT - (in reality there is no need for a train to be in section 1BT

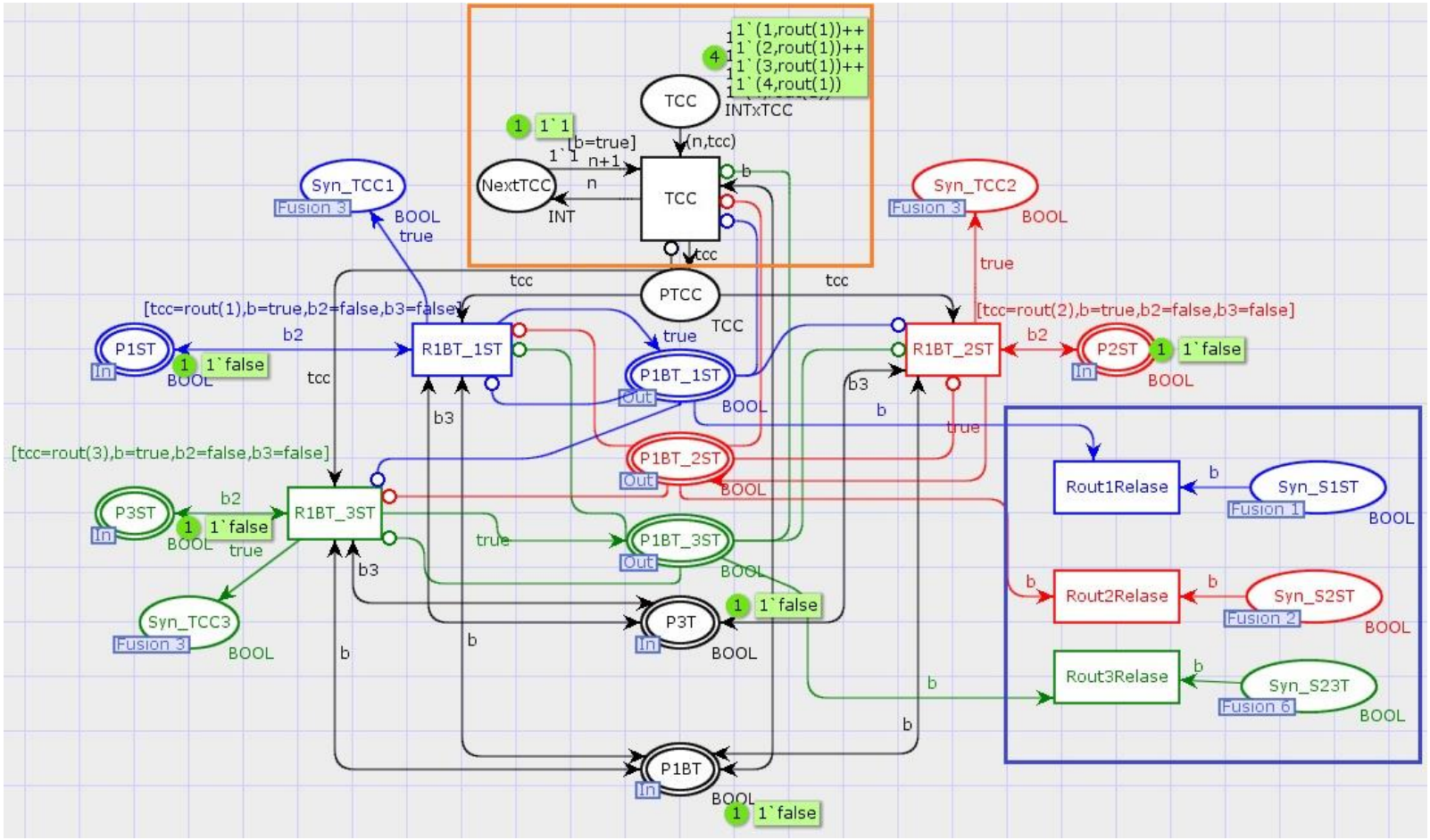


Figure 4.5: Interlocking model.

but this condition is considered here for simulation purposes). Elements surrounded by blue are responsible for route releasing process which will be explained later.

After a request is issued from TCC to reserve a certain route, based on the requested route, a group of parameters is needed to be checked. In 1BT-2ST case, transition "R1BT-2ST" is responsible for reserving 1BT-2ST route. In case that no route is reserved, a train is in section 1BT, suitable request for route reservation, section 3T and 2ST are empty, transition "R1BT-2ST" will put "true" on place "P1BT-2ST" as an indication that the rout is reserved. Figure 4.6 depicts the sequence of route reservation.

For routes 1BT-1ST and 1BT-3ST to be reserved, the same procedures will be followed, but the elements needed to be checked will be different.

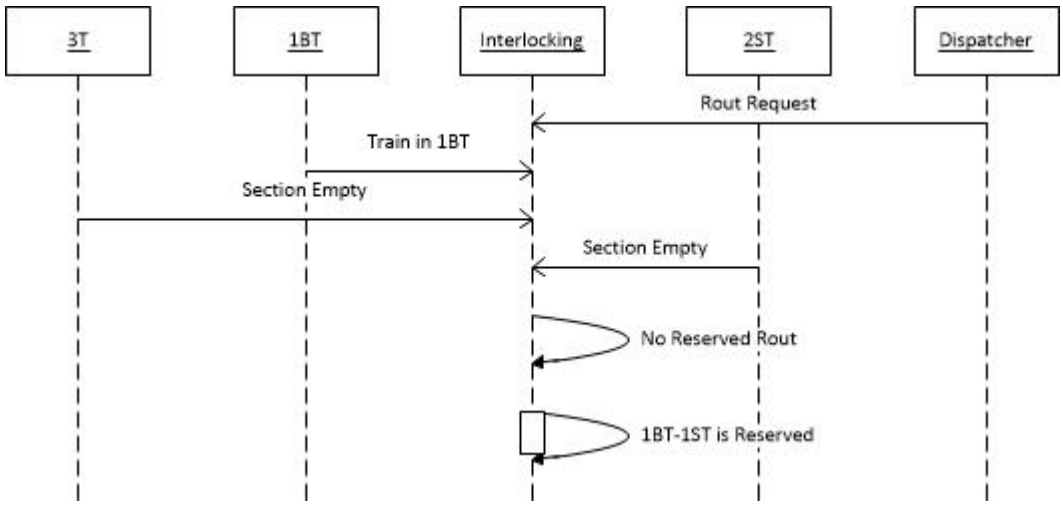


Figure 4.6: 1Bt-2ST route reservation sequential diagram.

4.1.4 Switches

After a rout is reserved, moving elements of train line must be set at certain positions according to the reserved route (In case of 1BT-2ST both SW1 and SW3 must be normal). Figure 4.7 shows SW1 model, it was proposed that switches works ideally. In general, switch must be either normal or reverse if there is no running process. Switch can be nether normal nor reverse just during the switching process for a certain time, but it can never be both in the same time. Switch 1 movement activated based on the reserved route. If 1BT_2ST or 1BT_3ST is reserved, it will be checked if the switch is in reversed position or not. If yes it will be moved to the normal place, if no it will be locked in this position. In case of 1BT_1ST the opposite will be

applied. Same procedures are applied for Switch 3, but it is just related to 1BT_2ST and 1BT_3ST.

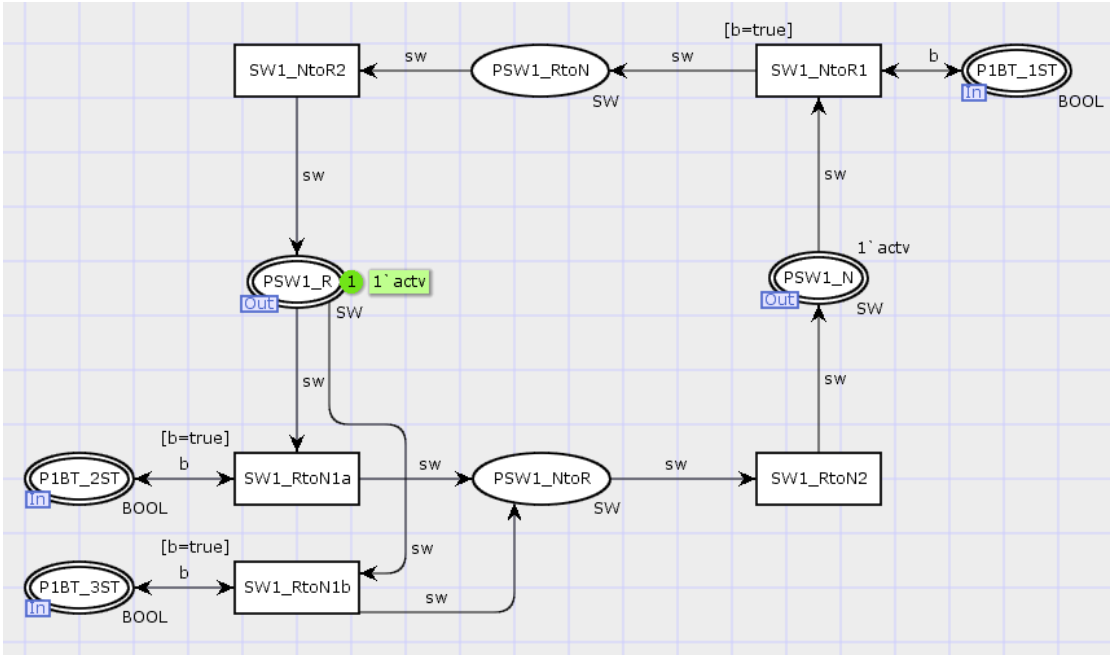


Figure 4.7: Switch-1 model.

After moving elements are locked to the appropriate positions, the train must be authorized to enter section 3T.

4.1.5 Section 3T and signal 2D

Signal 2D is responsible for authorizing the train to enter section 3T (Figure 4.8). As signal 2D is related to three lines, it was modeled in a separated module. Signal 2D module is interfaced with interlocking, switches, next signal for every line and to place “App_3T”. Signal 2D module is shown in Figure 4.9. A group of elements must be checked before authorization. As we are dealing with 1BT_2ST route, the module will check the following conditions:

1. Rout 1BT_2ST is reversed.
2. Switches 1 and 3 are at normal position.

If these conditions are satisfied, based on the next signal color of the related section Signal 2D color will be changed (Table 3.1). Supposing that signal 52DA is yellow-green, Figure 4.10 shows the Sequential Diagram of changing the color of signal 2D from red to yellow.

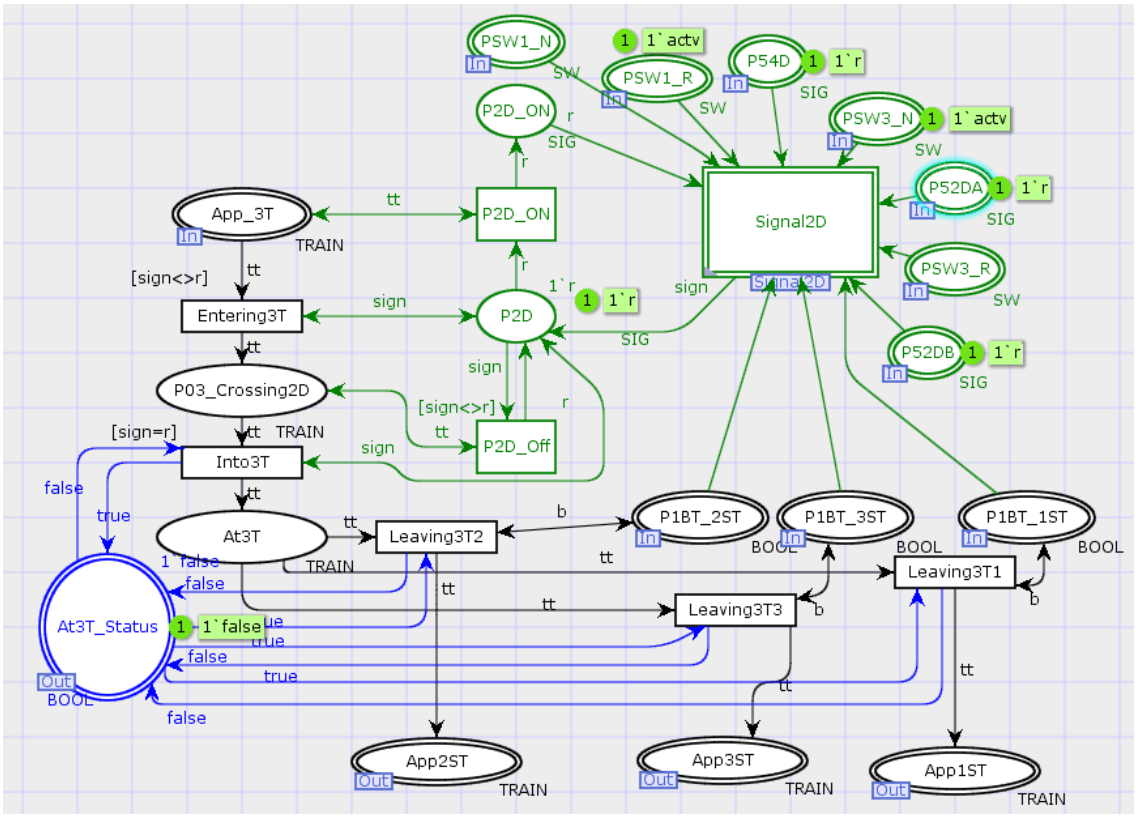


Figure 4.8: Section 3T model.

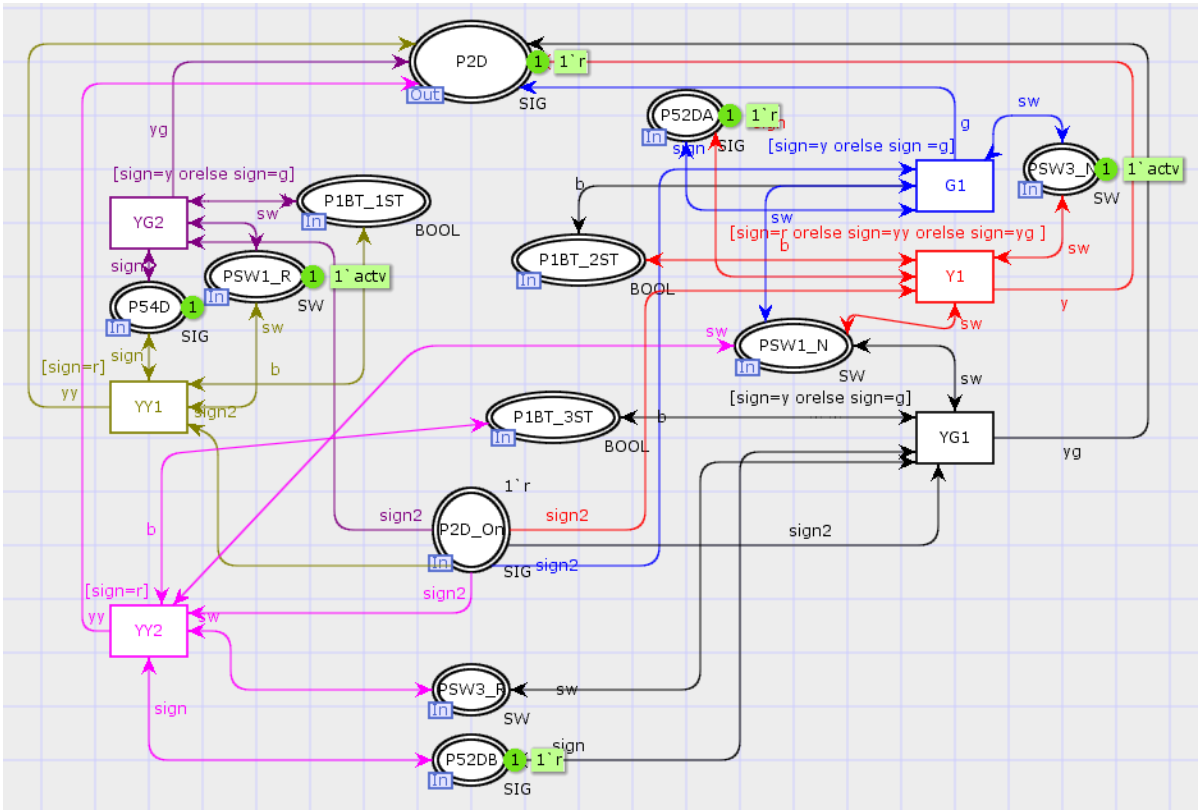


Figure 4.9: Signal 2D model

When 2D authorizes for train to enter, train can enter 3T section. In the moment train get into section 3T, Signal 2D becomes red again and token on place “At3T_Status” becomes “true”. Based on the reserved route, the train will take a path from the exit places of module S3T. On 1BT_2ST route case, token will pass through transition “Leaving3T2” and replacing the token on place “At3T_Status” to “false” again. By leaving section 3T, train enters section 2ST (Figure 4.11).

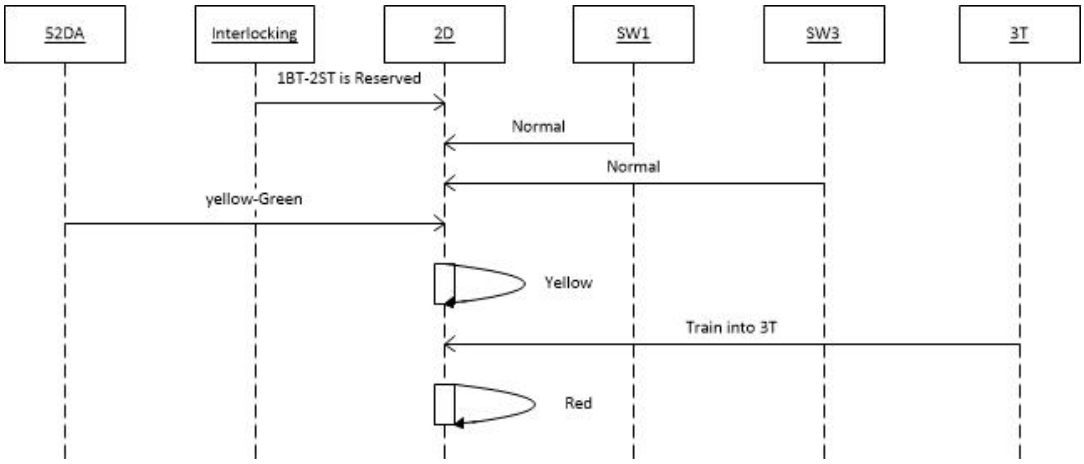


Figure 4.10: Signal 2D sequential diagram.

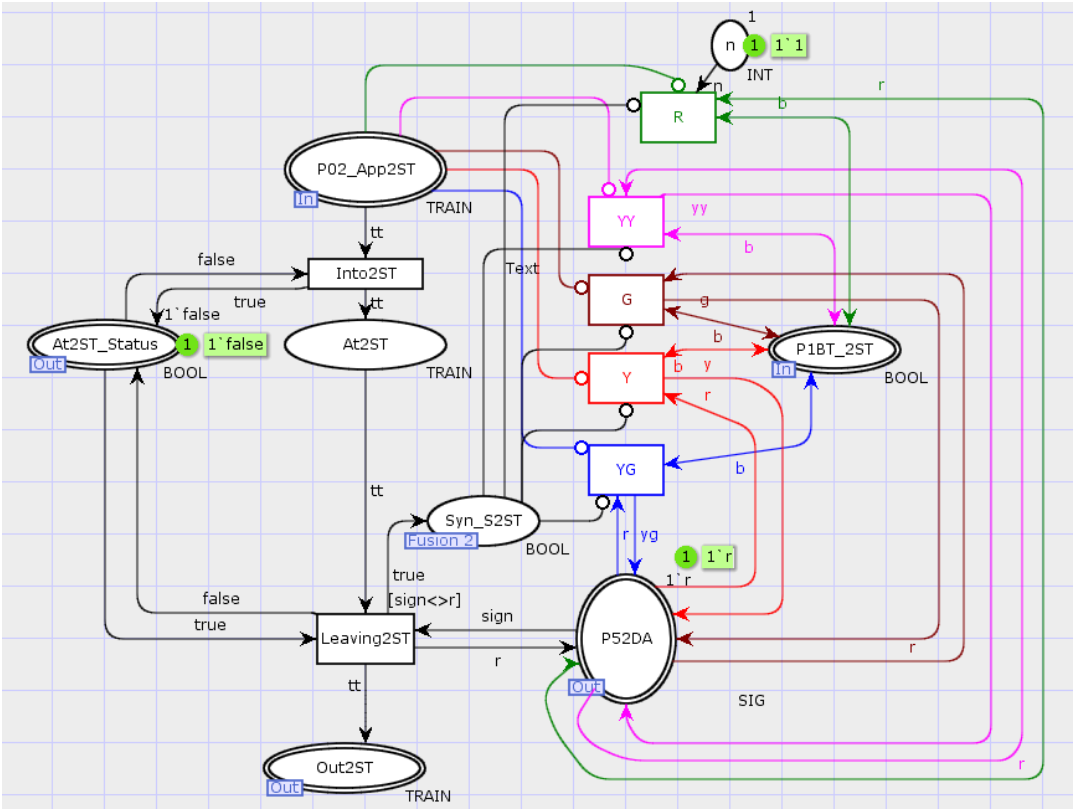


Figure 4.11: Section 2ST model.

4.1.6 Section 1ST, 2ST and 3ST

Train needs extra authorization to enter section 2ST from section 3T, as 2D is the signal that protects the whole targeted yard. The colored transitions in Figure 4.11 are responsible for changing signal 52DA color. As model is just concerning with the yard in Figure 3.1, signal 52DA is changed randomly when 1BT_2ST route is reserved (same thing for 52DB and 54D). When the train gets into place “At2ST”, token on place “At3T_Status” becomes “true”. While train is on place “At2ST” there is a possibility that Signal 52DA is red. In this case, train will wait and in the mean while transitions responsible for changing signal 52DA are enabled and will be fired randomly. This process will keep on till signal 52DA is not red. This process is virtual process, not for realization. This process can continue infinitely, so transition “R” in S1ST was provided with place “n” which has one token. This place will prevent generating red color over signal 52DA more than once. It is just modeled to simulate the waiting of the train when the signal is red. When signal 52DA is not red, train can pass and reserved route can be released.

4.1.7 Route releasing and fusion sets

To understand releasing mechanism using fusions, fusions must be reviewed first.

Till now tokens moved between modules using ports and sockets. Fusion sets enable places in different modules to be joined together into one compound place across the hierarchical structure of the model [30].

Fusion sets were used in order to release routes. For this purpose 4 sets were used (Figure 4.5, Figure 4.11 and Figure 4.12). In 1BT_2ST case, after train is leaving 1ST section, a “true” token will be generated over place “Syn_S1ST”. Because it is fusion with place “Syn_S1ST” in interlocking module, the same token will be generated on place “Syn_S1ST” in interlocking module. By moving to interlocking module it will be noticed that transition “Rout2Relase” is enabled due to the existence of token in place “Syn_S1ST” and place “P1BT_2ST”. By firing transition “Rout2Relase”, tokens on both places will be lose causing 1BT_2ST to release. Same rule is applied for other routes.

Fusion 3 set is used to prevent train token to leave section 1BT if a route is not reserved (Figure 4.3 and Figure 4.5**)**. Such a procedure cannot be realized, because section 1BT departure is something related to driver`s commitment of movement

laws. But it was used here to reduce the number of enabled transitions at the same time, because multi enabled transitions at the same time cause random order of firing which increases the number of state spaces.

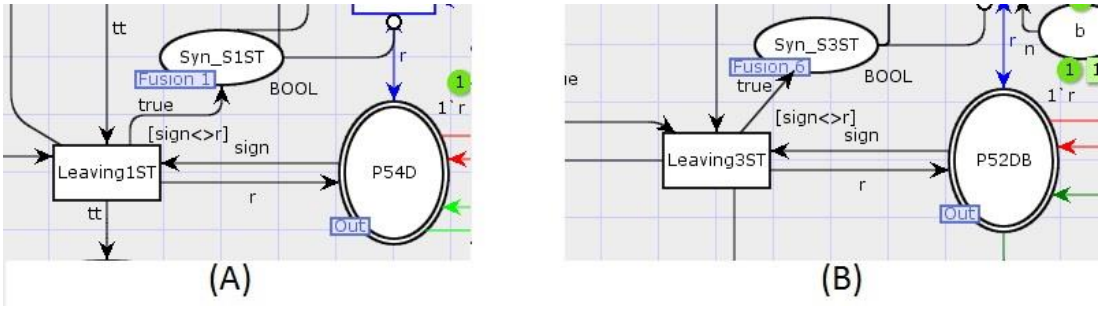


Figure 4.12: (A) 1ST fusion sets, (B) 3ST fusion sets.

According to simulation, system is working properly. Next section shows the procedures of making sure that system does not contain any liveness and fairness.

4.2 Verification

State Space will be used to verify the modeled system. Before a state space can be calculated, it is necessary to generate the state space code and to draw SCC graph. By generating a state space report, a report like in Figure 4.13 is resulted.

The report was generated by proposing that 6 trains are passing using the 3 routes. Figure 4.13 shows part of the state space report for the system. The system has 3205 nodes/markings and 6985 arcs in space state and Scc graph. According to [29], if state space and Scc have the same number of nodes and arcs means that the system has no cycles. System has no home marking that can be reachable from any marking that can be reached from initial marking. System has 8 dead markings. Those dead markings cannot be classified as safe or not safe without checking it. Because the existence of dead markings is something normal in this case study as all tokens of trains will stop in some places so the system will stop. The report shows that the system has 1 dead transition which is used to change SW1 position from reverse to normal when a request reservation for 1BT_3ST route is delivered. This transition is not used because when the report was generated, 1BT_3ST route request came after 1BT_3ST route request so there was no need to change the position of SW1 through that transition.

```
Statistics
------------------------------------------------

  State Space
     Nodes:  3205
     Arcs:   6985
     Secs:   1
     Status: Full
  Scc Graph
     Nodes:  3205
     Arcs:   6985
     Secs:   0
Home Properties
------------------------------------------------
  Home Markings
     None
Liveness Properties
------------------------------------------------

  Dead Markings
     8 [3205,3203,3202,3201,3196,...]

  Dead Transition Instances
     SW1'SW1_RtoN1b 1

  Live Transition Instances
     None

Fairness Properties
------------------------------------------------
     No infinite occurrence sequences.
```

Figure 4.13: System state space report.

Figure 4.13 shows that the system has no fairness too. To check if the dead lock is safe or not, query in Figure 4.14 was used. The query shows that all the 8 dead markings are considered safe.

This query works by defining what is the safe marking. It was defined here by the places (with “TRAIN” colour) as empty places. The only places excluded from these conditions are place “Out1ST”, place “Out2ST” and place “Out3ST”, because trains are supposed to leave all sections at the end. The markings that were classified as dead marking will be checked if they are safe according to the criteria that were defined. Figure 4.15 shows that all the dead markings were classified as safe dead markings. Figure 4.16 and Figure 4.17 show the queries that were used to check the existence of livelocks and self-loops respectively. Both of the queries confirm that system has no livelocks and self-loops.

System was verified that it is working properly. Now it needs to be validated to check if it preforms the required tasks or not.

```
fun ValidTerminal n=(Mark.S1BT'Depo 1 n=[] andalso
Mark.S1BT'P02_App1BT 1 n=[] andalso
Mark.S1BT'P03_CrossingB2D 1 n=[] andalso
Mark.S1BT'At1BT 1 n=[] andalso
Mark.S1BT'App3T 1 n=[] andalso
Mark.S3T'P03_Crossing2D 1 n=[] andalso
Mark.S3T'At3T 1 n=[] andalso
Mark.S3T'App2ST 1 n=[] andalso
Mark.S3T'App3ST 1 n=[] andalso
Mark.S3T'App1ST 1 n=[]andalso
Mark.S1ST'At1ST 1 n=[] andalso
Mark.S2ST'At2ST 1 n=[] andalso
Mark.S3ST'At3ST 1 n=[]  )
fun InvalidTerminal()=PredNodes(ListDeadMarkings(),fn n=>not(ValidTerminal n),NoLimit);
let
   val fid=TextIO.openOut"DeadlockMarkings.txt"
   val _ = TextIO.output(fid,"Not safe deadlock markings:\n")
   val _ = EvalNodes(InvalidTerminal(),fn n=>INT.output(fid,n))
   in
   TextIO.closeOut(fid)
end;
```

Figure 4.14: Safe dead markings query.

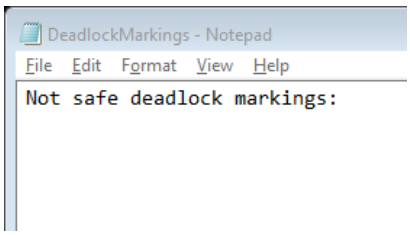


Figure 4.15: Safe dead markings query result.

```
   fun ListTerminalSCCs()=PredAllSccs(SccTerminal);
   fun InvalidTermSCC()=PredSccs(ListTerminalSCCs(),fn n=>not(SccTrivial n),NoLimit);
   let
      val fid=TextIO.openOut"AbsenceOfLiveLocks.txt"
      val _ = if InvalidTermSCC()=[]
      then TextIO.output(fid,"No Livelocks!")
      else  TextIO.output(fid,"yes Livelocks!")
        in
      TextIO.closeOut(fid)
   end;
```

Figure 4.16: Livelocks checking query.

```
fun SelfLoopTerminal n=(OutNodes(n)=[n])
fun InvalidTerminal()=PredNodes(EntireGraph,fn n=>(SelfLoopTerminal n),NoLimit);
let
   val fid=TextIO.openOut"ListOfSelfLoops.txt"
   val _ = TextIO.output(fid,"self terminals markings:\n")
   val _ = EvalNodes(InvalidTerminal(),fn n=>INT.output(fid,n))
   in
   TextIO.closeOut(fid)
end;
```

Figure 4.17: Self loops checking query.

4.3 Validation

In section 3.2 were mentioned. In this section these criteria are going to be tested and validated. Not all of the criteria will be explained here, because some of them work in the same mechanism but with changes in the input Appendix C shows the whole queries.

To check if no more than one train can be in 1BT at the same time, query in Figure 4.18 was used.

```
fun UnexpectedMarking n =(Mark.S1BT'P02_App1BT 1 n =[tr(1)] andalso
Mark.S1BT'At1BT 1 n =[tr(2)] );
val myASKCTLformula=INV(NOT(NF(" No Than one Train can be in 1BT",UnexpectedMarking)));
eval_node myASKCTLformula InitNode;
```

Figure 4.18: First criterion checking query.

This query works by checking if there is any marking that have more than one train in the places of the section 1BT. But is it applied just for two places at the same time so it must be repeated between all the places of the section 1BT. If the system satisfies the first criterion, a result like Figure 4.19 will appear. If not the "true" surrounded by red will be "false".

```
val UnexpectedMarking = fn : Node -> bool
val myASKCTLformula =
  NOT
    (EXIST_UNTIL (TT,NOT (NOT (NF (" No Than one Train can be in 1BT",fn)))))
  : A
val it = true : bool
```

Figure 4.19: First criterion checking query result.

To see if B2D is not giving permission if a rout is reserved or there is another train in 1BT, query in Figure 4.20 is used. It works by defining the proper initial conditions for every place related to signal B2D and defining the final condition of B2D place. As long as the conditions are true, the result will be "true" (the same in Figure 4.20). To confirm that signal B2D does not authorize to the train to enter in improper conditions, the proper initial conditions were changed and the result was "false". Initial conditions were changed continuously to check all the possible probabilities. Figure 4.21 shows the result of the second criterion with wrong initial conditions.

```
fun StartMarking n =(Mark.S1BT'P03_CrossingB2D 1 n =[] andalso
Mark.S1BT'At1BT 1 n =[] andalso
Mark.S1BT'B2D 1  n=[r] andalso
Mark.TCC'P1BT_1ST 1 n=[] andalso
Mark.TCC'P1BT_2ST 1 n=[] andalso
Mark.TCC'P1BT_3ST 1 n=[] );
fun EndMarking n =(Mark.S1BT'B2D 1  n=[g]);
val myASKCTLformula=POS(AND(NF("it can happen",StartMarking),
      EXIST_UNTIL((NF("",StartMarking)),
                  NF(" P03 to P02", EndMarking))));
eval_node myASKCTLformula InitNode;
```

Figure 4.20: Second criterion checking query.

```
val StartMarking = fn : Node -> bool
val EndMarking = fn : Node -> bool
val myASKCTLformula =
  EXIST_UNTIL
    (TT,
     NOT
       (OR
         (NOT (NF ('it can happen',fn)),
          NOT (EXIST_UNTIL (NF ("",fn),NF (" P03 to P02",fn)))))) : A
val it = false : bool
```

Figure 4.21: Second criterion checking query result with wrong initial conditions.

The rest of the queries will be shown in Appendix C.

4.4 Machine Performance

The process of modeling, validation and verification were done using a machine with the specifications shown in Figure 4.22

System

Processor:	Intel(R) Core(TM) i5 CPU M 460 @ 2.53GHz 2.53 GHz
Installed memory (RAM):	4.00 GB (3.80 GB usable)
System type:	64-bit Operating System, x64-based processor
Pen and Touch:	No Pen or Touch Input is available for this Display

Figure 4.22: Machine specifications.

While building the state space model of the system, as it can be seen in Figure 4.23, the process occupies about 62 MB from the memory and about 29% of the CPU in 32 second.

cpntools (32 bit) (2)	28.2%	61.9 MB	0 MB/s	0 Mbps

Figure 4.23: System performance

5. CONCLUSION

In this work, railway signalling system verification and validation using CPN was demonstrated. It is shown that it is easy to build a system model using CPN. The verification and validation processes were easy too, as they are computationally not expensive (see 4.4). So, if the model was generated, it does not take long time and effort to scan the whole markings of the system.

Checking the marking is not easy using just CPN tools. Because the current CPN tools are not developed enough so you need to generate all of the markings in a manual way. Alternatively, you can use some tools which are little complex in building the markings of the model (like Graphviz - Graph Visualization Software).

The disadvantage of this method is that it does not guarantee the completeness of the system. Only the stated properties are being checked. So, it is the developer`s responsibility to state the whole important properties of the system in order to be checked.

CPN with CTL* could validate an existing petri net system, but do not generate a PLC code for this system. It must be seen if it is possible to generate plc code based on the CPN model directly.

REFERENCES

[1] **Peterman, David Randall, and William J. Mallet.** "The Federal Role in Rail Transit Safety". Congressional Research Service, 2009.

[2] "Meeting the standard-engineering acceptance." RAILWAY SAFETY-PAPERS FROM THE RAILWAY TECHNOLOGY CONFERENCE HELD AT RAILTEX 2000, NATIONAL EXHIBITION CENTRE, BIRMINGHAM, UK, 21-23 NOVEMBER 2000. 2001.

[3] **Anders, E.,** "Railway signalling & interlocking: international compendium". Eurailpress, 2009.

[4] **Cook, S. R.** "Railway signalling-achieving concurrent safety and reliability." RAILWAY SAFETY-PAPERS FROM THE RAILWAY TECHNOLOGY CONFERENCE HELD AT RAILTEX 2000, NATIONAL EXHIBITION CENTRE, BIRMINGHAM, UK, 21-23 NOVEMBER 2000. 2001.

[5] **ERİŞ, O.** "BİR DEMİRYOLU ANKLAŞMAN SİSTEMİNİN PLC İLE GERÇEKLENMESİ", (Master Thesis), (2011).

[6] **Durmuş, Mustafa Seçkin**. "A CONTROL AND AUTOMATION ENGINEERING APPROACH TO RAILWAY INTERLOCKING SYSTEM DESIGN", (Ph.D. Thesis), (2014).

[7] **Jansen, H., and H. Schäbe**. "Computer architectures and safety integrity level apportionment." Safety and Security in Railway Engineering (2010): 19.

[8] **Stephens, E. J.** "C580/120/2000 Seeing the light." IMECHE CONFERENCE TRANSACTIONS. Vol. 5. Professional Engineering Publishing; 1998, 2001.

[9] **Durmuş, Mustafa Seçkin, et al.** "Synchronizing automata and Petri net based controllers." Electrical and Electronics Engineering (ELECO), 2011 7th International Conference on. IEEE, 2011.

[10] **Belmonte, Fabien, et al.** "Role of supervision systems in railway safety."Safety and Security in Railway Engineering (2010): 59.

[11] International Electrotechnical Commission. "Functional safety of electrical/electronic/programmable electronic safety related systems." IEC 61508 (2000).

[12] EN50126, C. E. N. E. L. E. C. "Railway application–The specification and demonstration of dependability, reliability, availability, maintainability and safety (RAMS)." (2000).

[13] EN50129, C. E. N. E. L. E. C. "Railway applications-Communication, signalling and processing systems-Safety related electronic systems for signalling." British Standards Institution, United Kingdom. ISBN (2003): 0580-4181.

[14] EN, BS. "50159: 2010 Railway applications–Communication, signalling and processing systems–Safety-related communication in transmission systems." М.: ФГУП «Стандартинформ (2010).

[15] EN, BS. "50128: 2011." Railway applications—Communication, signalling and processing systems—Software for railway control and protection systems, BSI Standards Publication (2011).

[16 **Dean, Neville**. “Teaching and learning formal methods”. Morgan Kaufmann, 1996.

[17] **Jackson, Daniel.** "Lightweight formal methods." FME 2001: Formal Methods for Increasing Software Productivity. Springer Berlin Heidelberg, 2001. 1-1.

[18] **R. W. Butler** *(2001-08-06).* "What is Formal Methods?". Retrieved 2006-11-16 http://shemesh.larc.nasa.gov/fm/fm-what.html

[19] **Almeida, José Bacelar, et al.** "An overview of formal methods tools and techniques." Rigorous Software Development. Springer London, 2011. 15-44.

[20] **Cassandras, Christos G., and Stephane Lafortune**. “Introduction to discrete event systems”. Springer Science & Business Media, 2009.

[21] **Durmuş, Mustafa Seçkin, and Mehmet Turan Söylemez**. "Automation Petri Net Based Railway Interlocking and Signalization Design." Int. Symposium on Innovations in Intelligent Systems and Applications. 2009.

[22] **Monin, Jean-François**. “Understanding formal methods”. Springer Science & Business Media, 2012.

[23] **Kropf, Thomas**. “Introduction to formal hardware verification”. Springer Science & Business Media, 2013.

[24] **Kelly, John C., et al**. "formal methods specification and verification guidebook for software and computer systems volume i: planning and technology insertion." NASA, July (1995).

[25] **Voros, Nikolaos S., Wolfgang Mueller, and Colin Snook**. "An Introduction to Formal Methods." UML-B Specification for Proven Embedded Systems Design. Springer US, 2004. 1-20.\

[26] **Cheng, Allan, Søren Christensen, and Kjeld Høyer Mortensen.** "Model checking Coloured Petri Nets-exploiting strongly connected components."DAIMI Report Series 26.519 (1997).

[27] **as Vojnar, Tom.** "Various kinds of petri nets in simulation and modelling."

(1997).

[28] **She, Xiaoli, Jiyuan Zhao, and Jian Yang.** "Functional safety verification on railway signaling system with Colored Petri Nets." Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on. IEEE, 2014.

[29] **Wu, Daohua, and Eckehard Schnieder.** "Scenario-based system design with colored Petri nets: an application to train control systems." Software & Systems Modeling (2016): 1-23.

[30] **Jensen, Kurt, and Lars M. Kristensen.** "Coloured Petri nets: modelling and validation of concurrent systems". Springer Science & Business Media, 2009.

[31] **Núñez, Manuel, et al., eds**." Applying Formal Methods: Testing, Performance, and M/E-Commerce" FORTE 2004 Workshops The FormEMC, EPEW, ITM, Toledo, Spain, October 1-2, 2004. Vol. 3236. Springer, 2004.

[32] **Katsaros, Panagiotis.** "A roadmap to electronic payment transaction guarantees and a Colored Petri Net model checking approach." Information and software technology 51.2 (2009): 235-257.

[33] **van Schuppen, J. H., M. Silva, and C. Seatzu.** "Control of discrete-event systems-Automata and Petri Net perspectives." Lecture notes in control and information sciences 433 (2013). p.249

[34] van der Aalst,W. "Fairness, Place Transition Invariants, and Siphons and Traps", Available at (http://cpntools.org) , (Reached on: 18.03.2016).

[35] **Jensen, Kurt, Søren Christensen, and Lars M. Kristensen.** "CPN tools state space manual." Department of Computer Science, Univerisity of Aarhus(2006).

[36] **Baier, Christel, and Joost-Pieter Katoen**." Principles of model checking". Vol. 26202649. Cambridge: MIT press, 2008.

[37] **Clarke, Edmund M., Orna Grumberg, and Doron Peled.** "Model Checking". MIT press, 1999.

[38] URL: https://en.wikipedia.org/wiki/Tree_structure (Reached on: 19.04.2016).

[39] URL: https://en.wikipedia.org/wiki/Computation_tree_logic (Reached on: 19.04.2016).

[40] **Christensen, Soren, and Kjeld H. Mortensen.** "Design/CPN ASK-CTL Manual." University of Aarhus (1996).

APPENDICES

Appendix A: How CPN works

Appendix B: Demonstration

Appendix C: Validation Queries

Appendix A

As it was mentioned before, CPN like any petri net consists of places (eclipses) and transitions (rectangles), the token is carried over the places, every place must have a color, and the token on the place must belong to the same color of the place, Figure A.1 shows a partial view from a net that was used in [20], It can be seen that “P01-OutRout”place color is defined as “TRAIN” and “p22-SigOff” place color is defined as “SIG”. “tr(1)” token belongs to “TRAIN” and “s” token belongs to “SIG”. “tr(1)” cannot be held by “SIG” and “s” cannot be held to by “P01-OutRout”.

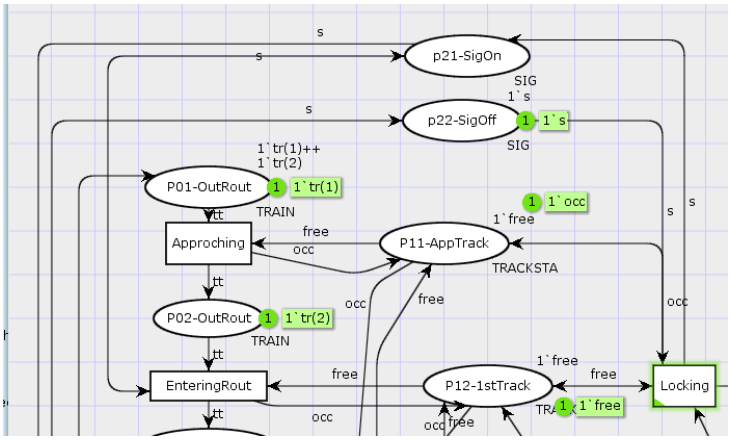


Figure A.1: Train yard model.

As it in petri nets, places and transition are connected through arcs, in CPN arcs must have inscriptions, which indicate which type of token is going to be transferred by arcs. For arcs going out of places, inscriptions can be either variable (which just moves the token in the place) or token name (which just accepts specified token from place color).

For arcs going out of transitions it is the same for the ones going out of places, but if the inscription is variable, the transition must have another arc as input which has the same inscription. In order to fire the transition all input arcs must be enabled. When the transition is enabled it becomes green in Figure A.1. “EnteringRout” transition is not enabled because “p21-SigOn” place does not have a token. “Locking” transition is enabled (it is green) and can be fired.

By firing “Locking” transition, the token on “p22-SigOff” place will be moved to “p21-SigOn” place (Figure A.2). With these conditions, “EnteringRout” transition is enabled and can be fired.

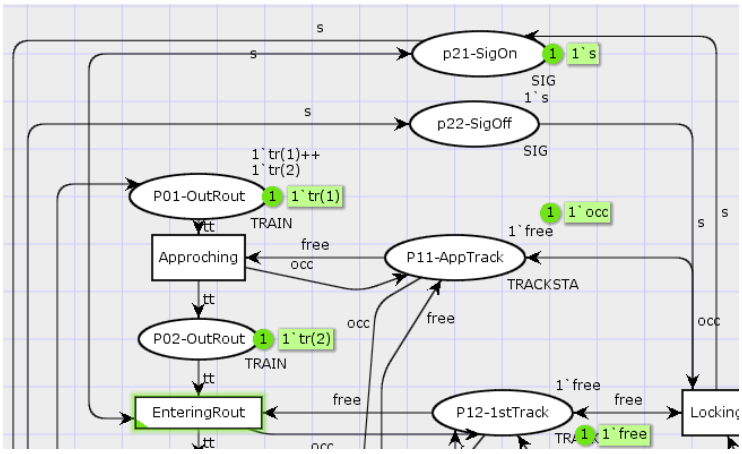


Figure A.2: Firing of "Locking" transition.

By firing "EnteringRout" transition, 3 moves will be done:

1. "p12-1stTrack" place will send "free" token and receive "occ" token.
2. The token on "p02-OutRout" place will be moved to "p04-CrossingSignal" place.
3. The token on "p21-SigOn" place will go and come back to the same place through "EnteringRout" transition (Figure A.3).

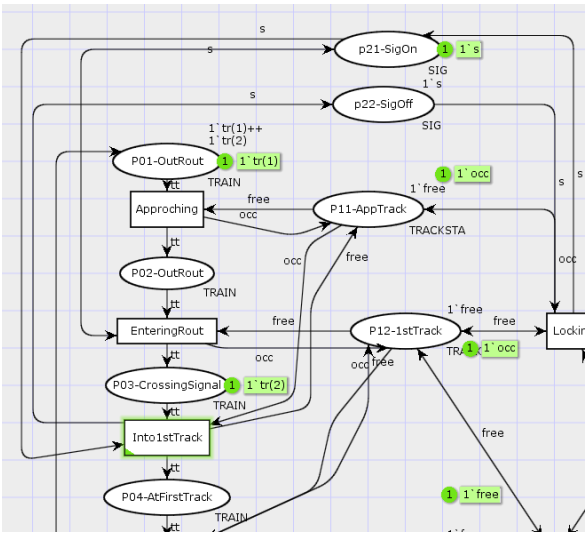


Figure A.3: Firing of "EnteringRout" transition.

Appendix B

The main model in [20] is used with small modifications (Figure B.1) to demonstrate the ability of CPN verification and validation.

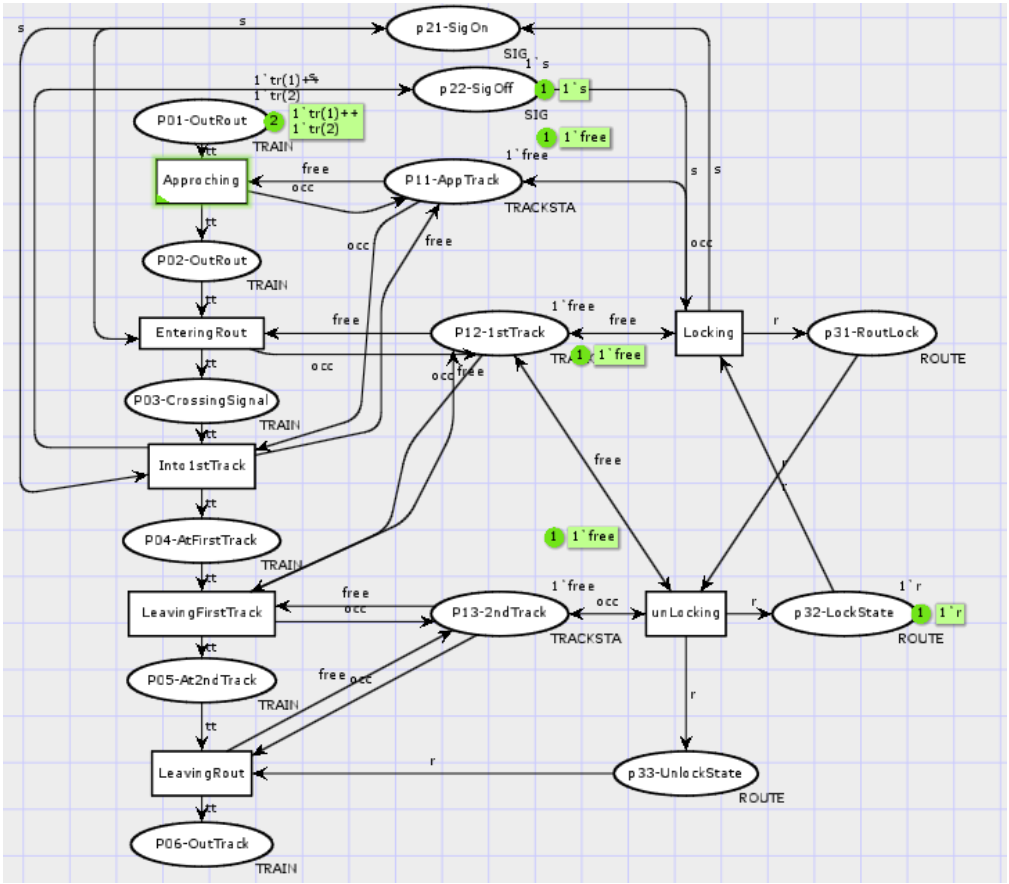


Figure B.1: Main CPN model 1.

In this model, it was proposed that there is a section in the track protected by a signal. Two trains will pass from this section. The rules of the section are that the two trains cannot enter the sections without permission from the signal, trains can be in the section at the same time and the trains are not allowed to go back through the section. By generating the state space nodes, arrangement in Figure B.2 is resulted. Node1 is the initial marking, there are two possibilities, tr(1) can pass first or tr(2) can pass first If tr(1) passes first marking2 is reached, but if tr(2) passes first marking3 is reached (Figure B.3). Based on every possibility, different scenarios can occur. Figure B.2 and Figure B.3 show the different scenarios, tr(1) is the first scenario which is surrounded by red and tr(2) is the first scenario which is surrounded by blue.

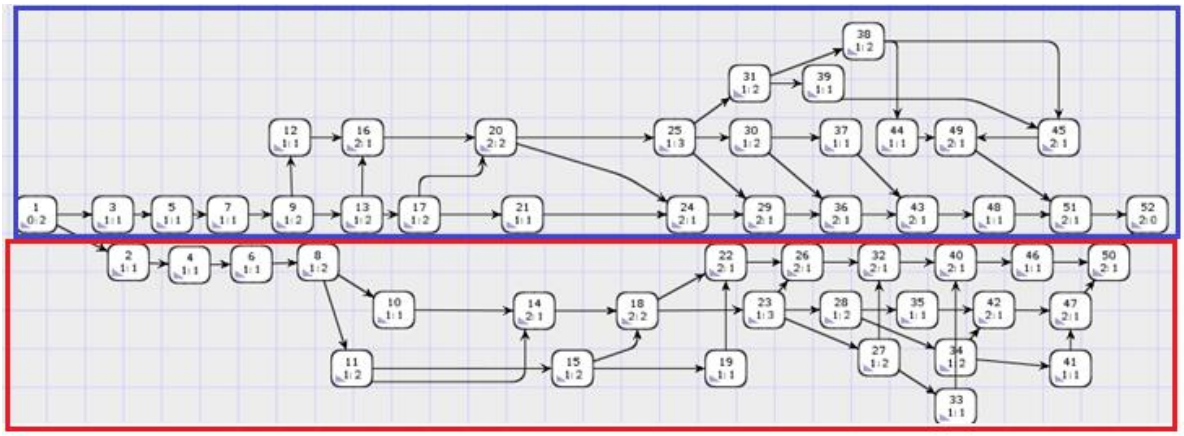


Figure B.2: State space nodes of model 1.

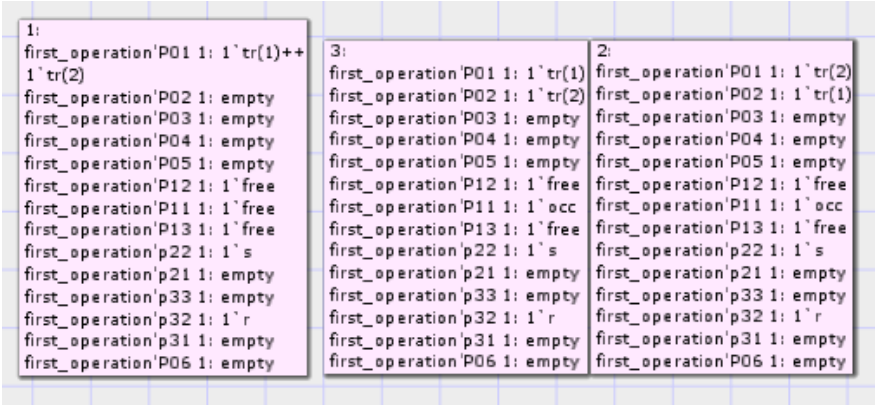


Figure B.3: Different marking of model1.

Figure B.4 shows the markings that occurred based on tr(2) First scenario. tr(1) cannot move till marking7, because the token is in P21 not in P22 (shown in red in Figure B.4). After reaching marking9, there are two enabled transitions- "Approaching" transition and "LeavingFirstTrack" transition. One of them will fired first randomly. After that, the process will be continued based on the random order that was applied (Figure B.5).

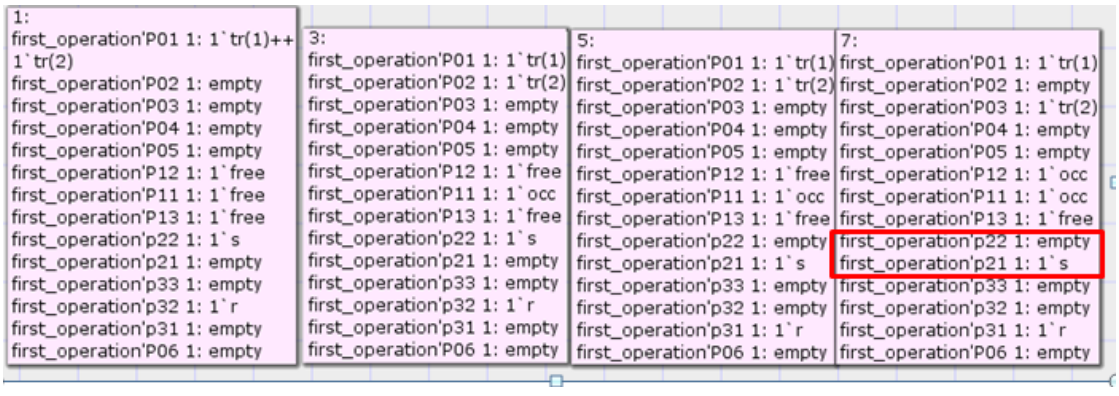


Figure B.4: tr(2) first scenario.

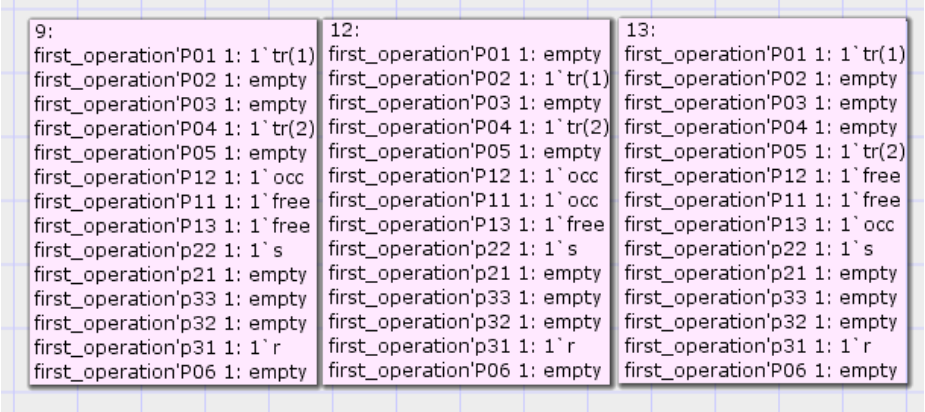


Figure B.5: Two enabled transition.

Figure B.6 shows part of the state space report of model1. The system has 52 nodes/markings and 70 arcs in space state and Scc graph. Node52 is home marking which means that it is reachable from any marking that can be reached from initial marking. Moreover node52 is dead marking. When node52 is reached, no other event can occur and this can be confirmed in Figure B.2.

```
Statistics
------------------------------------------------
  State Space
     Nodes:  52
     Arcs:   70
     Secs:   0
     Status: Full

  Scc Graph
     Nodes:  52
     Arcs:   70
     Secs:   0
 Home Properties
------------------------------------------------
  Home Markings
     [52]

 Liveness Properties
------------------------------------------------
  Dead Markings
     [52]

  Dead Transition Instances
     None

  Live Transition Instances
     None
 Fairness Properties
------------------------------------------------
     No infinite occurrence sequences.
```

Figure B.6: Part of the state space report of model1.

Model1 must have one deadlock marking when both trains leave the section. To make sure that the system reaches a safe deadlock, next query was used.

(*Query 1*)

fun ValidTerminal n=(Mark.fo'P01 1 n=[] andalso

```
Mark.fo'P02 1 n=[] andalso
Mark.fo'P04 1 n=[] andalso
Mark.fo'P03 1 n=[] andalso
Mark.fo'P05 1 n=[]  )
fun InvalidTerminal()=PredNodes(ListDeadMarkings(),fn n=>not(ValidTerminal n),NoLimit);
let
  val fid=TextIO.openOut"DeadlockMarkings.txt"
  val _ = TextIO.output(fid,"List of deadlock markings:\n")
  val _ = EvalNodes(InvalidTerminal(),fn n=>INT.output(fid,n))
  in
  TextIO.closeOut(fid)
end;
```

In this query, the criterion of the safe deadlock was defined, so it examines all the deadlock of the system then shows the unsafe deadlock. For model1 it was found that there is no unsafe deadlock.

To examine if the system has self-loop next query was implemented, and it showed that there is no self-loop, which means that safe termination cases are included in the list of dead markings.

```
(*Query2*)
fun SelfLoopTerminal n=(OutNodes(n)=[n])
fun InvalidTerminal()=PredNodes(EntireGraph,fn n=>(SelfLoopTerminal n),NoLimit);
let
  val fid=TextIO.openOut"ListOfSelfLoops.txt"
  val _ = TextIO.output(fid,"List of self terminals:\n")
  val _ = EvalNodes(InvalidTerminal(),fn n=>INT.output(fid,n))
  in
  TextIO.closeOut(fid)
end;
```

To examine if the system has livelocks, next query was implemented, and it showed that system has no livelocks.

```
(*Query3*)

fun ListTerminalSCCs()=PredAllSccs(SccTerminal);
fun InvalidTermSCC()=PredSccs(ListTerminalSCCs(),fn n=>not(SccTrivial
n),NoLimit);
let
  val fid=TextIO.openOut"AbsenceOfLiveLocks.txt"
  val _ = if InvalidTermSCC()=[]
  then TextIO.output(fid,"No Livelocks!")
  else  TextIO.output(fid,"yes Livelocks!")
   in
  TextIO.closeOut(fid)
end;
```

Till now the mentioned queries verify the system and show that system can work probably. To validate the system ASK-CTL queries are going to be used. The first property to be checked is that train will not go back. Next query show if the train can go back between any two points. And for Model1 this property is validated. This query must be implemented between every two points in the section.

```
(*Query4*)

fun StartMarking n =(Mark.fo'P02 1 n =[tr(1)]);

fun EndMarking n =(Mark.fo'P03 1 n =[tr(1)]);

val myASKCTLformula=POS(AND(NF("olmuyor",StartMarking),

    EXIST_UNTIL((NF("",StartMarking)),

            NF(" P03 to P02", EndMarking))));

eval_node myASKCTLformula InitNode;
```

Last restriction to be validated, no more than one train can be in the section at the same time, next query was used for this purposes, all the possible probabilities was

examined. The query shows that it is not possible for more than one train to be in the section at the same time.

(*Query5*)

fun UnexpectedMarking n =(Mark.fo'P02 1 n=[tr(1)] andalso

Mark.fo'P03 1 n=[tr(2)]);

val myASKCTLformula=INV(NOT(NF("olmuyor",UnexpectedMarking)));

eval_node myASKCTLformula InitNode;

The queries show that the system is working properly. To make sure that that the queries are working properly, model 1 was modified by adding two exit places to the system, “P07” and “P08” model2 was resulted (Figure B.7)

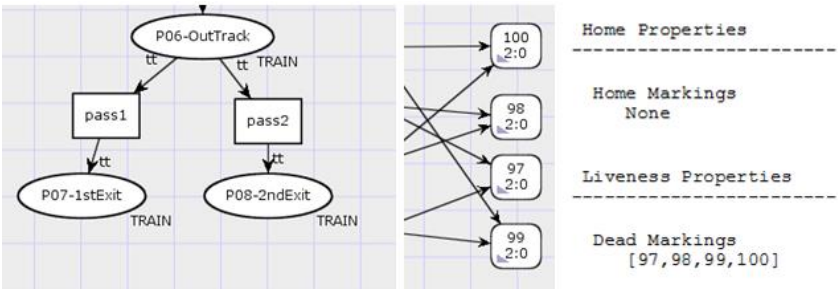


Figure B.7: Model2.

It can be seen from the space state nodes that there are 4 possible dead markings for model2, space state report shows the number of deadlock markings space state nodes. It is noticed from Figure B.8 that all the dead markings have the same settings apart from the last two places. It can be seen too that model2 has no home markings.

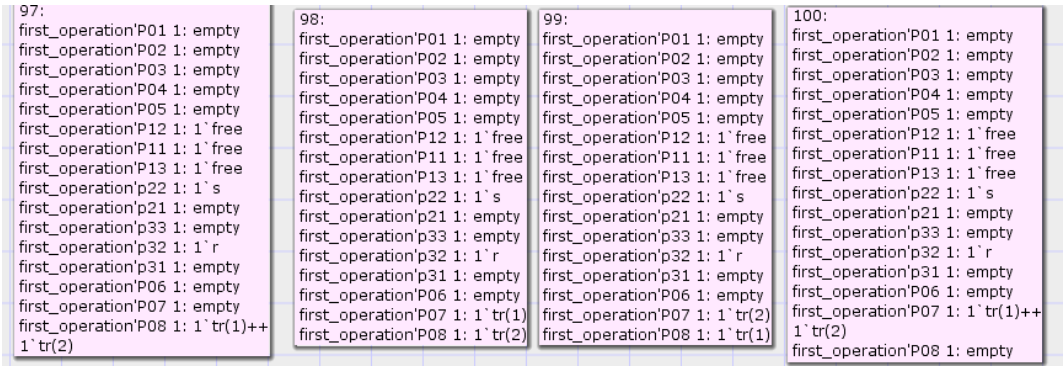


Figure B.8: Model2 deadlock markings state space nodes.

Qurey1 was used to prove that those deadlock markings are safe – some changes where applied to the query to fit with model2. It is not necessary to test other query on model2 because it has no big difference from model1. So new model were produced (model3) by modifying model1 by adding a loop to the end of the net as shown in Figure B.9.

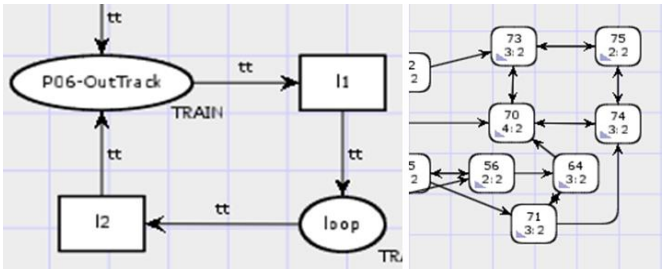


Figure B.9: Model3.

Figure B.10 shows that the number of state space nodes and arcs is not the same of Scc Graph`s, which indicates that there is a cycle\s in model3, from Figure B.9 the existence of cycles can be approved, It can be seen too that the nodes involved in the cycle are the same in Figure B.9 and in Figure B.10, the system has no deadlock markings, but it has 4 home markings- same markings involved on the cycle. Query3 was implemented to model3 in and it emphasized that the model3 has livelocks.

```
------------------------------------------------
  State Space
     Nodes:  75
     Arcs:   138
     Secs:   0
     Status: Full
  Scc Graph
     Nodes:  52
     Arcs:   90
     Secs:   0
 Home Properties
------------------------------------------------
  Home Markings
     [70,73,74,75]
 Liveness Properties
------------------------------------------------
  Dead Markings
     None
  Dead Transition Instances
     None
  Live Transition Instances
     first_operation'l1 1
     first_operation'l2 1
 Fairness Properties
------------------------------------------------
  Impartial Transition Instances
     first_operation'l1 1
     first_operation'l2 1
  Fair Transition Instances
     None
  Just Transition Instances
     None
  Transition Instances with No Fairness
     first_operation'Approching 1
     first_operation'EnteringRout 1
     first_operation'Into1stTrack 1
     first_operation'LeavingFirstTrack 1
     first_operation'LeavingRout 1
     first_operation'Locking 1
     first_operation'unLocking 1
```

Figure B.10: Model3 space state report.

Model4 was generated by adding a cycle in the middle of model1 (Figure B.11), the area surrounded by red is the modified part, when train comes to P04 a random rout will be followed, either by entering the loop or by staying on the normal track. Figure B.12 shows model4 state space report, from the report the existence of cycle\s can be figured out easily, by examining the state space graphs, the existence of livelocks loops can be easily observed too. From the results, the ability of CPN tools to model, verify and validate the system was demonstrated.

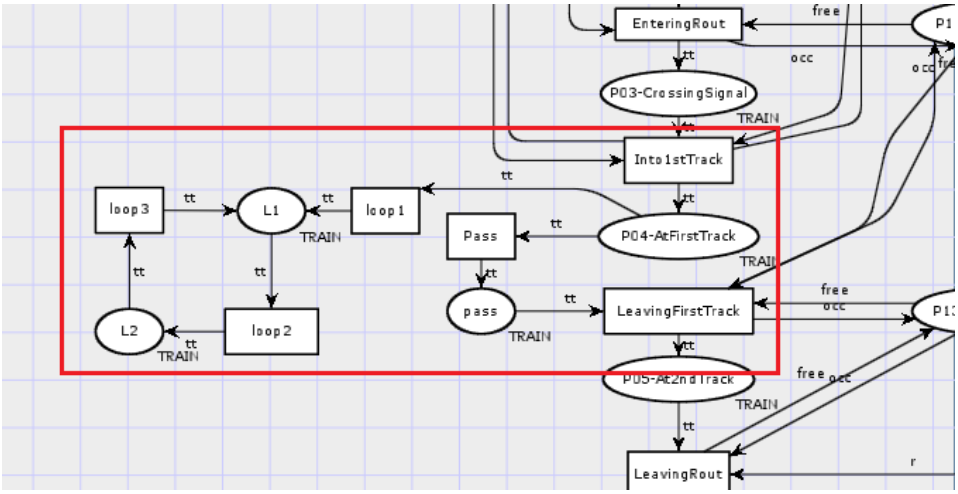


Figure B.11: Model4.

```
Statistics
------------------------------------------------
  State Space
     Nodes:  88
     Arcs:   136
     Secs:   0
     Status: Full
  Scc Graph
     Nodes:  76
     Arcs:   112
     Secs:   0
Home Properties
------------------------------------------------
  Home Markings
     None
 Liveness Properties
------------------------------------------------
  Dead Markings
     [88]
  Dead Transition Instances
     None
  Live Transition Instances
     None
 Fairness Properties
------------------------------------------------
  Impartial Transition Instances
     first_operation'loop2 1
     first_operation'loop3 1
  Fair Transition Instances
     first_operation'EnteringRout 1
     first_operation'Into1stTrack 1
     first_operation'LeavingFirstTrack 1
     first_operation'Locking 1
     first_operation'Pass 1
     first_operation'loop1 1
     first_operation'unLocking 1
  Just Transition Instances
     None
  Transition Instances with No Fairness
     first_operation'Approching 1
     first_operation'LeavingRout 1
```

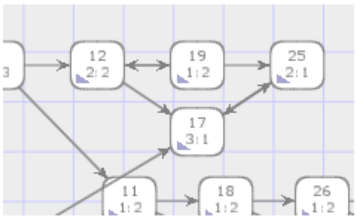


Figure B.12: Model4 space state report and graph.

Appendix C

The rules were mentioned in 3.2 will be written again followed by its query.

1. No more than one train can be in 1BT at the same time.

```
fun UnexpectedMarking n =(Mark.S1BT'P02_App1BT 1 n =[tr(1)] andalso
  Mark.S1BT'At1BT 1 n =[tr(1)] );
val myASKCTLformula=INV(NOT(NF(" No Than one Train can be in
1BT",UnexpectedMarking)));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```

2. B2D is not giving permission if a rout is reserved or there is another train in 1BT.

```
fun StartMarking n =(Mark.S1BT'P03_CrossingB2D 1 n =[] andalso
Mark.S1BT'At1BT 1 n =[] andalso
Mark.S1BT'B2D 1  n=[r] andalso
Mark.TCC'P1BT_1ST 1 n=[true] andalso
Mark.TCC'P1BT_2ST 1 n=[] andalso
Mark.TCC'P1BT_3ST 1 n=[] );
fun EndMarking n =(Mark.S1BT'B2D 1  n=[g]);
val myASKCTLformula=POS(AND(NF("it can happen",StartMarking),
    EXIST_UNTIL((NF("",StartMarking)),
            NF(" P03 to P02", EndMarking))));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```

3. No train leaves 1BT without a reserved rout.

```
fun UnexpectedMarking n =( Mark.S1BT'At1BT 1 n =[tr(1)] andalso
Mark.TCC'P1BT_1ST 1 n=[] andalso
Mark.TCC'P1BT_2ST 1 n=[] andalso
Mark.TCC'P1BT_3ST 1 n=[]  );
val myASKCTLformula=INV(NOT(NF(" it cannot
happen",UnexpectedMarking)));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```

4. For 1BT_1ST to be reserved, none of the routs must be reversed, a train must be on section 1BT, section 1ST must be free, section 3T must be free and request to reserve 1BT_1ST from TCC must be delivered.

```
fun StartMarking n =(Mark.S1BT'At1BT_Status 1 n =[true] andalso
Mark.S1ST'At1ST_Status 1 n =[false] andalso
Mark.TCC'P1BT_1ST 1 n=[] andalso
Mark.TCC'P1BT_2ST 1 n=[] andalso
Mark.TCC'P1BT_3ST 1 n=[true] andalso
Mark.TCC'PTCC 1 n=[rout(1)]  );
fun EndMarking n =(Mark.TCC'P1BT_1ST 1 n=[true] );
```

```
val myASKCTLformula=POS(AND(NF("it can happen",StartMarking),
    EXIST_UNTIL((NF("",StartMarking)),
            NF(" P03 to P02", EndMarking))));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```

5. For 1BT_2ST to be reserved, none of the routs must be reversed, a train must be on section 1BT, section 2ST must be free, section 3T must be free and request to reserve 1BT_2ST from TCC must be delivered.

```
fun StartMarking n =(Mark.S1BT'At1BT_Status 1 n =[true] andalso
Mark.S2ST'At2ST_Status 1 n =[false] andalso
Mark.TCC'P1BT_1ST 1 n=[] andalso
Mark.TCC'P1BT_2ST 1 n=[] andalso
Mark.TCC'P1BT_3ST 1 n=[] andalso
Mark.TCC'PTCC 1 n=[rout(1)]  );
fun EndMarking n =(Mark.TCC'P1BT_2ST 1 n=[true] );
val myASKCTLformula=POS(AND(NF("it can happen",StartMarking),
    EXIST_UNTIL((NF("",StartMarking)),
            NF(" P03 to P02", EndMarking))));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```

6. For 1BT_3ST to be reserved, none of the routs must be reversed, a train must be on section 1BT, section 3ST must be free, section 3T must be free and request to reserve 1BT_3ST from TCC must be delivered.

```
fun StartMarking n =(Mark.S1BT'At1BT_Status 1 n =[true] andalso
Mark.S3ST'At3ST_Status 1 n =[false] andalso
Mark.TCC'P1BT_1ST 1 n=[] andalso
Mark.TCC'P1BT_2ST 1 n=[] andalso
Mark.TCC'P1BT_3ST 1 n=[] andalso
Mark.TCC'PTCC 1 n=[rout(3)]  );
fun EndMarking n =(Mark.TCC'P1BT_3ST 1 n=[true] );
val myASKCTLformula=POS(AND(NF("it can happen",StartMarking),
    EXIST_UNTIL((NF("",StartMarking)),
            NF(" P03 to P02", EndMarking))));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```

7. No train can enter 3T without permission from 2D.

```
fun StartMarking n =( Mark.S3T'P02_App3T 1 n =[tr(1)] andalso
  Mark.S3T'P2D 1 n =[r]);
fun EndMarking n =(Mark.S3T'P03_Crossing2D 1 n =[tr(1)]);
val myASKCTLformula=POS(AND(NF("it can happen",StartMarking),
    EXIST_UNTIL((NF("",StartMarking)),
            NF(" P03 to P02", EndMarking))));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```

8. In 1BT_2ST reservation case, for 2D to give permission, SW1 and SW3 must be normal, 1BT_2ST rout must be reserved and 52DA is not red.

```
fun StartMarking n =(Mark.SW1'PSW1_N 1 n =[actv] andalso
Mark.SW3'PSW3_N 1 n =[actv] andalso
Mark.TCC'P1BT_2ST 1 n=[true] andalso
Mark.S3T'P2D_ON 1 n=[r]  );
fun EndMarking n =(Mark.S3T'P2D 1 n=[y]
orelse Mark.S3T'P2D 1 n=[g] );
val myASKCTLformula=POS(AND(NF("it can happen",StartMarking),
    EXIST_UNTIL((NF("",StartMarking)),
            NF(" P03 to P02", EndMarking))));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```

9. In 1BT_1ST reservation case, for 2D to give a permission, SW1 must be reversed, 1BT_1ST rout must be reserved and 54D is not red.

```
fun StartMarking n =(Mark.SW1'PSW1_R 1 n =[actv] andalso
Mark.TCC'P1BT_1ST 1 n=[true] andalso
Mark.S3T'P2D_ON 1 n=[r]  );
fun EndMarking n =(Mark.S3T'P2D 1 n=[yy]
orelse Mark.S3T'P2D 1 n=[yg] );
val myASKCTLformula=POS(AND(NF("it can happen",StartMarking),
    EXIST_UNTIL((NF("",StartMarking)),
            NF(" P03 to P02", EndMarking))));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```

10. In 1BT_3ST reservation case, for 2D to give permission, SW1 must be normal, SW3 must be reversed, 1BT_3ST rout must be reserved and 52DB is not red.

```
fun StartMarking n =(Mark.SW1'PSW1_N 1 n =[actv] andalso
Mark.SW3'PSW3_R 1 n =[actv] andalso
Mark.TCC'P1BT_3ST 1 n=[true] andalso
Mark.S3T'P2D_ON 1 n=[r]  );
fun EndMarking n =(Mark.S3T'P2D 1 n=[yy]
orelse Mark.S3T'P2D 1 n=[yg] );
val myASKCTLformula=POS(AND(NF("it can happen",StartMarking),
    EXIST_UNTIL((NF("",StartMarking)),
            NF(" P03 to P02", EndMarking))));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```

11. No train can enter 3T if there is another train in the section.

    (Next rules will be applied for 1ST, 2ST and 3ST).

```
fun UnexpectedMarking n =(Mark.S3T'P02_App3T 1 n =[tr(2)] andalso
  Mark.S3T'At3T 1 n =[tr(1)] );
```

```
val myASKCTLformula=INV(NOT(NF(" it can not
happen",UnexpectedMarking)));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```

12. While train is going out of 3T section, it must assure that the section that the train is going to, is cannot be entered if the appropriate rout is not reserved.

```
fun UnexpectedMarking n =(Mark.S3T'App1ST 1 n =[tr(2)] andalso
  Mark.TCC'P1BT_1ST 1 n =[true] );
val myASKCTLformula=INV(NOT(NF(" it can not
happen",UnexpectedMarking)));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```

13. For 52DA to give permission, 1BT_2ST rout be reserved.

```
fun StartMarking n =(Mark.S2ST'P52DA 1 n =[r] andalso
Mark.TCC'P1BT_2ST 1 n=[] );
fun EndMarking n =(Mark.S2ST'P52DA  1 n=[yy]
orelse Mark.S2ST'P52DA  1 n=[g]
orelse Mark.S2ST'P52DA  1 n=[y]
orelse Mark.S2ST'P52DA  1 n=[yg]);
val myASKCTLformula=POS(AND(NF("it can happen",StartMarking),
    EXIST_UNTIL((NF("",StartMarking)),
            NF(" P03 to P02", EndMarking))));
eval_node myASKCTLformula InitNode;
----------------------------------------------------------------------
```