AUTOMATIC GENERATION AND VERIFICATION OF INDIAN RAILWAY INTERLOCKING CONTROL TABLES

By

K. SRIRAM (312214405033)

A Project Report

Submitted to the

Faculty of Information and Communication Engineering

in partial fulfillment of the requirements

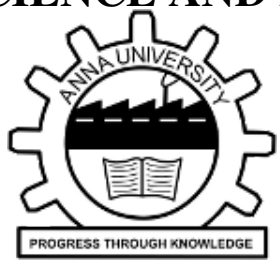
for the award of the degree

of

MASTER OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



ANNA UNIVERSITY

CHENNAI – 25

JUNE 2016

BONAFIDE CERTIFICATE

Certified that this project report titled **AUTOMATIC GENERATION AND VERIFICATION OF INDIAN RAILWAY INTERLOCKING CONTROL TABLES** is the *bonafide* work of K. SRIRAM (312214405033) who carried out the project work under my supervision, for the fulfillment of the requirements for the award of the degree of Master of Engineering in Software Engineering. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on these or any other candidates.

Place: Chennai
Date:

Dr. S. Sheerazuddin
Associate Professor
Department of Computer Science and Engineering
SSN College of Engineering

COUNTERSIGNED

Head of the Department
Department of Computer Science and Engineering
SSN College of Engineering
Kalavakkam 603 110

Submitted for the examination held on .....

Internal Examiner

External Examiner

ABSTRACT

Generating control tables for Interlocking system in Railways is a complex task. This is generally done by the vendor, who manufactures the hardware for Indian Railways. These control tables are verified for correctness by another vendor. The problem that exists with this current method is the process followed to generate these control tables remains a black box $(i.e.)$ unknown. The contribution to be made through this project is to explore this unknown process and come up with a system, that will give the correct control tables for a given layout of Railway section. For the verification of consistent routes in control table, modeling of signaling system for the given section has been done using Symbolic Model Verification (SMV) language and verify the SMV description using NuSMV tool.

திட்டப்பணிச் சுருக்கம்

இருப்புப்பாதை போக்குவரத்தில் பின்னிப்பூட்டல் அமைப்பிற்கு கட்டுப்பாடு அட்டவணைகளை உருவாக்குவது சிக்கலான பணி. இந்தப் பணியை, பொதுவாக, இந்திய இருப்புபாதை போக்குவரத்து துறைக்கு வன்பொருளை தயாரித்து தரும் விற்பனையாளர்களே செய்வார்கள். இவ்விற்பனையாளர் உருவாக்கித் தரும் கட்டுப்பாடு அட்டவணைகளை சரி பார்க்க மற்றுமொரு விற்பனையாளரிடம் அளிக்கப்படுகின்றது. இந்தக் கட்டுப்பாடு அட்டவணைகளை உருவாக்கவும், சரி பார்க்கவும் கையாளப்படும் செயல்முறை ஒரு கேள்விக்குறியாக இருப்பது தான் பிரச்சனை. இந்த மறைந்திருக்கும் செயல்முறையை அறியவும், சரியான கட்டுப்பாடு அட்டவணையை உருவாக்கவும், எடுக்கப்படும் இம்முயற்சியே, இந்தத் திட்டத்தின் வாயிலாக அளிக்கப்படவிருக்கும் பங்களிப்பாகும். கட்டுப்பாடு அட்டவணைகளில் முரணற்ற தடங்களின் கலவைகளை சரி பார்க்க சமிக்ஞை அமைப்பின் மாதிரியை Symbolic Model Verification (SMV) மொழியைக் கொண்டு செய்யப்பட்டுள்ளது. இதன் பின்னர் NuSMV-யைக் கொண்டு, SMV விளக்கத்தை சரி பார்க்கப்பட்டுள்ளது.

ACKNOWLEDGEMENT

I express my thanks to my Project Supervisor **Dr. S. Sheerazuddin**, Assistant Professor and Project Coordinator **Dr. R. S. Milton**, Professor, Department of Computer Science and Engineering, for their support and guidance.

I would like to express my heartier thanks to **Dr. Chitra Babu**, Professor and Head, Department of Computer Science and Engineering, **other faculty members and technical staff** for providing all facilities and support to meet my project requirements.

I express my sincere thanks to **all the Project Review Committee members**, for their timely advice.

I express my gratitude to the Principal, **Dr. S. Salivahanan and the management** of SSN College of Engineering for providing the resources and support to carry out the project successfully.

It gives me immense pleasure to express heartfelt thanks to **my parents, beloved friends and GOD** for helping me all the time.

K. Sriram

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

BACKGROUND AND MOTIVATION

When a train enters or leaves a railway station, it is important to be sure that it does not derail and does not collide with another train. Therefore, rules have to be made for when a train can enter and leave a station. Like other railway enterprises, Indian Railways uses interlocking systems for ensuring that the safety rules are respected. Such systems are deployed for enforcing these rules on the physical objects of the stations. For instance, the track segments must be aligned correctly in position to make the train to move either straight or turn.

1.1 INTRODUCTION

A railway station or a railway section is represented as a layout diagram. This layout diagram consists of track segments connected with one another, along with signals, points and level crossings. Signals convey the information regarding operation of train or the track on which the train is set to move, to the driver. Points are the intersection of two track segments. It is used to turn a moving train from main line to loop line. Level Crossings are the control gates that co-ordinate the movement between road transport and railway without collision.

Verification tools can be classified into *interactive* and *automatic* tools. Theorem provers, at the current state of art, are not fully automatic for their usual tasks. They are driven through user interaction. As a consequence, experts are needed that are familiar with logic and the particular proof system that underlies the prover. The proof task turns out to be very time and cost intensive if it can be completed at all. Model checking tools, in contrast, do

not provide a proof of correctness but rather execute an exhaustive search for errors in the state space of a model. It is an exhaustive test over all possibilities. This search can be done fully automatically. As a result, the user gets an answer that the checked requirement is either satisfied in the model or violated and, in this case, an example shows in which situation the violation may happen.

Many contributions have been made from researchers across the globe with respect to Model checking of Railways interlocking system. But their work is aligned to the Railways System in their country. As the Railways signaling system varies from country to country, it cannot be generalized for all countries. In India, there has been no publications in this field. This is due to the complexity of model checking and unpublished research work for this system. This project was suggested by Ex. Director/Indian Railways Prof. (Dr.) V Purnachandra Rao.

The objectives of this project are listed below:

1. Formal representation of railway section recognizable by a program.
2. Generation of control table entries for the railway section.
3. Verification of control table entries for safety conditions.

1.2 ORGANIZATION OF REPORT

This report is organized into discussion of Indian Railway's signaling system and interlocking system basics, various model checking techniques, different approaches to verify interlocking system, basics of NuSMV and about implementation of this project before concluding.

We explain the signaling system and interlocking system of Indian Railways in detail through chapter 2. In chapter 3, model checking is explained

in detail. A detailed analysis of different techniques handled to verify different countries' interlocking system is given in chapter 4. NuSMV is explained in detail in chapters 5 respectively. In chapter 6, implementation details with case study is given and we conclude in chapter 7.

CHAPTER 2

RAILWAYS SIGNALLING & INTERLOCKING SYSTEM

In this chapter, the concepts involved in signalling and interlocking of Indian Railways are to be discussed in detail. The different types of signals and its meaning are discussed before having a look at interlocking concepts and its types.

2.1 SIGNALLING CONCEPTS

Railway vehicles move on steel rail track and are provided with flanged steel wheels. The rolling of steel wheel on steel rail has the least friction and it is, therefore, one of the most efficient means of locomotion.

2.1.1 Control over movement of trains

Running of flanged vehicles on the steel track has its own inherent problems unlike the road, sea or air transport where the movement is not confined to a particular track. Since the vehicles are constrained to move in a fixed railway track, they cannot be steered away as in the case of other transports. They are required to follow one another in the same direction on the length of track, as otherwise for every vehicle separate parallel paths are to be provided. This is not practical. If vehicles are expected from the opposite direction another set of diversion track is required to be provided either for overtaking vehicles moving in the same direction or for crossing the vehicles from the opposite direction. Railway locomotion, therefore, though more efficient, brings in problems of "control over movement of trains".

Basically, two types of controls could be catered for. If two separate tracks are provided for trains running in opposite directions, then one set of

control can be provided to space the movement of trains running in the same direction so that adequate "interval" is available between two consecutive trains. On the other hand, if a single track is used for movement of trains in both directions, then another set of control is required to prevent a train in the opposite direction from coming on the same track when a train is already occupying it.

2.1.2 Time Interval Method

Let us take the first case of spacing of trains in the same direction. The spacing should be such that if a train stops, then, the following train driver should be able to notice it and apply brake to his train so that it stops short of the preceding train. The most important aspect is bringing to a stop from the speed at which a train is running. Where the speeds and weights are low, it is not difficult for a following train to stop short of the train ahead, which has stopped. This is how tramway operate even today, as the speed and weight are low and a tram can be stopped from its running speed without colliding with a tram in front. With higher speeds and heavier loads, as in the case of train, the distance required to stop a train is longer, and at this longer distance, the driver cannot definitely decide whether a train in front has actually stopped or not. This is the case when trains follow one another in quick succession. In actual practice, where interval between trains is longer, a following train does not see the earlier train, and the driver has to continuously guess as to where the earlier train will be. If all trains run at the same speed and are required to stop at the same place for the same duration, a certain amount of control can be exercised by having a definite time lag between the trains from one stopping place to another. This time lag should be such that the train, which has a stop, is able to reach the next stop within this time. Thus by having a time interval between trains, a certain amount of control can be

achieved. But, in the case of Railway, this is not practicable. A better method of control is called the "Space Interval Method" is adopted.

2.1.3 Space Interval Method

In this method of "Control over movement", the length of track is divided into sections called "Blocks". The entry of a train into the block is controlled in such a way that only when it is free, a train can be allowed to enter it. This means that between two consecutive trains, there is a definite space interval.

This space interval or block is controlled at the entry. This controlling point should know whether the train, which had entered this space, vacated it so that another following train can be sent. Since the length of a block is beyond the normal visual range, another controlling point is set at the end of the block. This point can know whether the train has arrived and advise the controlling point at the entry. So, with the two controlling points and intercommunication, it is possible to control the entry of a train into a block only when it is vacant.

The information about the condition of this block is given by the exit point to the entry point, and the entry point transmits this information to the driver of a train. The driver of the approaching train must be able to know whether the next block is not clear, he should stop and wait. Here is where "signal" comes in to picture.

2.1.4 Signals

A "Signal", therefore, is a medium to convey a particular pre-determined meaning in non-verbal form. Various methods are used to convey the meaning by "signals" in a non-verbal form as are used by Scouts, Policemen, road signs, Navy and Air Traffic Control, etc., which convey a definite informa-

tion. The chart given in Figure 2.1, gives the various forms that could be adopted.

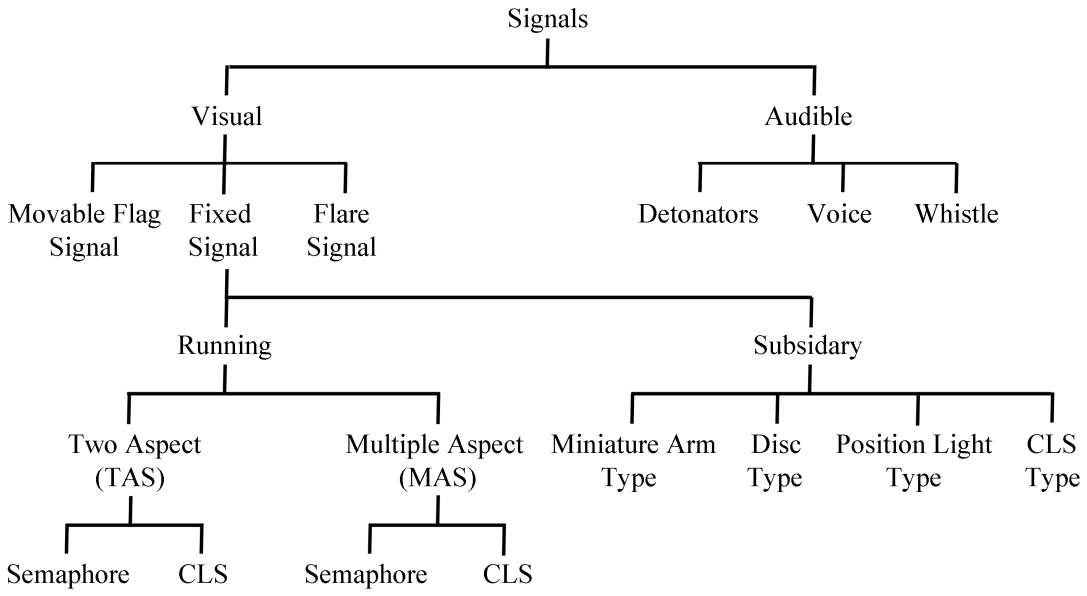


Figure 2.1 Various Forms of Signals

2.1.5 Block Working

As explained earlier, the space interval system uses the block working wherein the entry of train onto the block section is jointly controlled by the entry and exist points of the block section. The driver is authorized to proceed into a section by the signal controlling the entry to the section. This working could be a manual block system or automatic block system. In any type before the train could be allowed to enter a section "PERMISSION" is required to be obtained from the exit end to the effect that the section is "CLEAR" of trains and the train could be permitted. Different systems of working for getting this "PERMISSION TO APPROACH" have been evolved on Indian Railways and are classified as "System of Working".

2.2 FIXED SIGNALS, ASPECTS & INDICATIONS

In signals, a mention was made about the use of different types of visual and audible signals, for controlling the movement of trains in all cases. No exceptions are allowed by approved special Instructions in the following:

(a) Fixed signals

(b) Hand signals

(c) Detonating signals

(d) Flare signals

The definition of "Fixed Signals" as given in the General Rules is "a signal of fixed location indicating a condition affecting the movement of a train and includes a semaphore arm or disc or fixed light for use by day and a fixed light for use by night".

Semaphore signals used on the Railways are in the form of a rectangular or fish tailed arm fixed to a vertical post. The arm is kept horizontal to the post to be easily distinguishable. By this arrangement the arm can be seen from a long distance on a clear day. Whenever the signal is required to convey some information the arm can altogether be removed from the view of the driver by making the arm to disappear in a slot provided on the post, or

(a) The arm can be made to assume a mid-way position below horizontal, or

(b) To assume a mid-way position above horizontal, or

(c) To assume a vertical position parallel to the extended line of the post.

Method (a) was adopted in the early days and subsequently given up as the absence of arm due to some reason other than its entering the slot in the post conveyed wrong information. Methods (b) and (c) above could be on the Right hand side or left hand side of a Quadrant as shown below in Figure

2.2. Fixed Signals can operate on any one of the four quadrants of a circle as shown. Since 'Left hand' rule is followed in India, the "lower quadrant" and "Upper quadrant" of the left hand side is utilized in Indian Railways. Based on this principle, signals are also generally located on the left side of the track.

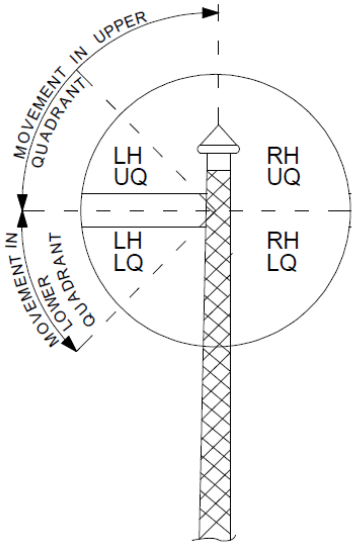


Figure 2.2 Use Of Quadrant

It can be seen in the Figure 2.2 that an arm in a lower quadrant can have only two positions, one at horizontal position and the other at midway position on the left-hand side. In Upper Quadrant, three positions can be obtained, i.e. one at horizontal position, one at midway position and the 3rd at vertical position in parallel with the extended line of post. Hence, we have two systems of signalling, one called "Lower Quadrant Signalling" and the other called "Upper Quadrant Signalling".

2.2.1 Two aspect Lower Quadrant Signalling

(a) Stop Signal The semaphore arm of the stop signal is square ended, painted red with white bar parallel to the square end in front and painted

white with black bar in rear. As explained, a lower quadrant signal can show only two different positions. One is horizontal and the other lowered to midway position. They are called “aspects” of the signals. The movement of the signal arm in lower quadrant is generally adopted by countries where there is no snowfall or other external conditions which can result in the arm remaining lowered without being operated. The arm in the horizontal position will convey an aspect “stop” indicating “Stop dead”. The arm lowered to midway position in the lower quadrant will convey an aspect “proceed”, indicating Proceed. Semaphore arm can be seen during day and so can convey information during daytime. At night the arm will not be visible. Hence, to convey information during night, fixed light signals are used. Right from the early days, red lights were used to denote “Stop” and green lights were used for “Proceed”. Red light should, therefore, be exhibited when the arm is horizontal and green light when the arm is inclined midway. A semaphore signal is a combined integrated unit with an arm and light. The horizontal position of the arm during daytime is considered as the “ON” aspect and the inclined position is the “OFF” aspect of the signal. The corresponding light red & green during night time are ‘ON’ and ‘OFF’ aspects respectively. The ‘ON’ aspect of a signal is also referred to as the most restrictive aspect.

(b) Warner Signal Two-aspect stop signal as explained above is the minimum required to safely space the trains. This is adequate for low speeds and low density of traffic. Safety depends on the driver seeing the signal in time under all conditions. This imposes an enormous strain on the driver who has to be constantly on the lookout, to pick up the signals. Any mistake or loss of attention can lead to serious consequences. Otherwise drivers will play safe by running at lower speeds so that he can stop at the signal even if he sees

it at the last minute. With such low speeds, the time of occupying the block sections by the trains will increase, thereby reducing the number of trains per day that can be run between the block stations.

One method of overcoming this problem will be to give information in advance, or "WARNING" to the driver about the presence of stop signal ahead and the aspect displayed by the stop signal. This can be achieved in the form of another signal. This signal can precisely inform the driver that he is approaching a stop signal and also that he is required to stop or proceed. The signal which gives such warning about the condition of the stop signal ahead is called a "WARNER SIGNAL".

Since the driver is not required to stop at the warner signal, as it is only giving an advance warning about the presence of the stop signal ahead, this signal has to be different from the stop signal. The day aspect, therefore, is characterized by a fish tailed arm instead of a square ended arm. This is also a two aspect lower quadrant signal.

Since the warner signal is not a stop signal and is exhibiting red light when 'ON this should be distinguishable from a stop signal during night. This is done by mounting the arm at a lower level in the post and providing a separate additional fixed green light at 1.5 to 2.0 meters above the arm. This combination of green light above a red light distinguishes a signal as a warner signal in the 'ON' position. When the signal is lowered to midway position, the red light changes to green and the driver sees two green lights one above the other. Two precise informations are given to the driver by the Warner Signal. When the arm is horizontal during day and showing of a green light and red light below during night time indicates to the driver that he can proceed, but must be prepared to stop at the next stop signal. Similarly, the lowering of arm during day and showing of two green lights one below the

other during night indicates that he can proceed and can expect all the stop signals ahead of warner for that direction are OFF and he can run through main line.

A warner signal must not be capable of being taken 'OFF' for any line other than that over which the highest speed is permitted (i.e. main line) and not until all the relevant signals have assumed 'OFF' aspect. The last of the stop signals will be the one controlling the entry of the train in the block section ahead. Even if any one of the stop signals ahead is 'ON' the warner cannot display 'OFF' aspect.

Under certain circumstances a semaphore warner signal is required to be placed on the same post of a stop signal. In such cases, the warner signal is placed below the stop signal, and the fixed green light is dispensed with.

The combination of two arms (stop and warner) on the same post gives the driver three indications in the 2-aspect lower quadrant signalling. When both the stop signal and the warner signal arms are at horizontal position and the showing of two red light one below the other gives an indication to the driver to 'stop dead' at this signal. The lowering of the stop signal above the warner or showing of a green light above a red light indicates that he can proceed past the signal with caution and be prepared to stop at the next stop signal. A third condition exists when both the arms are lowered to give two green lights one below the other. This indicates to the driver that he can proceed and can expect all the stop signals for that direction are 'OFF' and that the block section ahead is also clear. It is also made mechanically impossible to lower only the warner signal when the stop signal above it is 'ON'. In this way showing of green light below a red light is eliminated.

From the point of view of the driver, therefore, the 'ON' aspect of warner does not signify positively anything about the signals ahead whereas if such

information is available, he can confidently approach the signal ahead. A system of warning about the condition of each signal by a signal in rear is, therefore, very much necessary. This leads to the concept of more than 2 aspects called “MULTIPLE ASPECT SIGNALING”.

2.2.2 Multiple Aspect Upper Quadrant Signalling

(a) Stop Signal It has been mentioned in previous para that the semaphore arm can be made to assume a midway position above horizontal and also another position in parallel with the extended line of the post on the left hand upper quadrant. In this way, it is possible to obtain more than 2 aspects in the upper quadrant region and hence, it is called “Multiple Aspect” (more than 2 aspects) “Upper Quadrant” signalling as distinct from “two aspect Lower Quadrant Signalling” mentioned in previous paras.

The raising of the semaphore arm to “45° above horizontal” in the left hand upper quadrant region will convey an aspect “Caution” indicating “Proceed with caution and be prepared to stop at the next stop signal”. The night aspect of the mid-way position by showing of a yellow light. The raising of the arm to 90° above horizontal in parallel with the extended line of post in a vertical position will convey an aspect “Clear” indicating “Proceed” and the next stop signal is also ‘OFF’. The corresponding night aspect is the showing of a green light.

(b) Distant Signal As discussed in the case of 2-aspect signalling when a driver approaches the first stop signal he should be warned about its condition. Therefore, a signal similar to the warner signal in the 2-aspect signalling is also a necessity in multiple aspect upper quadrant signalling. This pre-warning signal is called a “DISTANT” signal. The term ‘Distant’ is used here, as this is the farthest signal from the station on the approach side. The

semaphore arm will have 3 positions - horizontal, 45° above horizontal and 90° above horizontal. The arm is fishtailed similar to lower quadrant warner signal. The front side facing the train is colored yellow with a black bar parallel to the end and the backside is colored white with a black bar.

The three positions of a multiple aspect upper quadrant both of a stop signal and a distant signal are horizontal for the 'ON' position, raised to 45° above horizontal and raised to 90° above horizontal are the 'OFF' positions.

So far we have discussed two types of signals i.e. Lower Quadrant 2-aspect and Multiple Aspect Upper Quadrant. The warner/distant signals are not stop signals and, therefore, "Permit" the approaching driver to pass the signal in the 'ON' position. Hence they are called "Permissive Signals". The stop signals in the 2 aspect and multiple aspect cannot be passed by the approaching driver in the 'ON' position unless and until he is specially authorized. Hence, these signals are called "Absolute Signals".

The above two types of semaphore signals are 2-aspect lower quadrant and 3-aspect upper quadrant whether permissive signals or absolute signals. The lights exhibited in the nighttime are lighted by "Kerosene Wick Lamps" or by electric lamps and they are lit only during the night time. In some areas, where the visibility of arm is very poor due to snow or fog, the night aspects are required to be lit in the day time also. The lighting of the lamps is left to the operating staff.

Multiple Aspect Color Light Signals Instead of having an arm by day and light by night it is preferable to have only lights as signals for both day and night and such signals are called color light signals. These are mainly used in busy suburban sections and main trunk routes, as these require electric power to operate them. Use of color light signals is essential in the electrified

sections.

At a block station it is obligatory to provide certain number of signals for controlling the movements of trains. There we require some signals to deal with the trains approaching the station and some to deal with departure of trains from the station. When more than one stop signals are used a difficulty to identify them from each other will arise. Hence it is necessary to give some name to these signals.

2.3 DESTINATION OF SIGNALS

At a block station it is obligatory to provide certain number of signals for controlling the movements of trains. There we require some signals to deal with the trains approaching the station and some to deal with departure of trains from the station. When more than one stop signals are used a difficulty to identify them from each other will arise. Hence it is necessary to give some name to these signals.

2.3.1 Signals for Reception

Signals, which are governing the approach and entry of trains into a station, are:

(a) Permissive signals A 'WARNER' in case of 2-Aspect signalling can be placed below the first stop signal or below the last stop signal or can be on a post by itself with fixed green light above. It is to warn the driver that he is approaching a stop signal or to warn him about the condition of block section ahead. In multiple aspect signalling a "DISTANT" signal is provided to indicate the driver about the condition of the stop signal ahead. If the sectional speed is 120 kmph or above, two "DISTANT" signals shall be provided. In such cases, these signals are called 'DISTANT' and 'INNER DISTANT' re-

spectively.

(b) Stop signals Minimum one permissive and one stop signal is sufficient for trains approaching a station. When stop signal is taken 'OFF' it permits the train to enter the station, this is called "HOME" signal of the station. At a station where two stop signals are provided in the approach, the first one shall be called 'OUTER' and the next shall be "HOME". In some cases where the distance between the home signal and the reception lines of the station is far away, one more stop signal may be provided, as one home signal will not be sufficient to facilitate the reception. So a stop signal provided between home and the reception lines shall be called a "ROUTING HOME".

2.3.2 Signals for Departure of Trains

At the departure end of the station, the stop signals controlling the movement of trains leaving the station are:

(a) Starter signal Where the departure of trains is controlled by only one stop signal, it is called starter signal and is the last stop signal of the station. If two or more converging lines are there, the starter shall be placed outside all connections on the line to which it refers. Where advanced starter is also provided, the starter referring to any line is placed so as to protect the facing point or fouling mark and shall not be less than 400m in advance of the Home signal.

(b) Advanced Starter Where departure of trains is controlled by more than one stop signal, the outer most starter signal shall be the last stop signal of the station and is called "Advanced Starter". Unless approved under special instructions an "Advanced Starter" shall be placed outside all connections

on the line to which it applies. It shall be placed at not less than 180m in the case of two aspect and 120m in multiple aspect signalling from the outermost point on single line and out side all connection. This distance shall be reckoned from the starter on double line. On special nominated sections where frequent shunting involving main line takes place the "Advanced Starter" signal may be placed at a distance of full train length beyond the trailing point and the track between trailing point and the advance starter shall be track circuited. Where an advanced starter is provided, the starter referring to any line shall be placed so as to protect the first facing point or fouling mark; and shall not be less than 400m in advance of home signal.

(c) Intermediate/Routing Starter Intermediate Starter is provided between starter & advanced starter where necessary, and is placed in rear of the point, which it protects.

We have seen the aspects and indications of an individual signal. The following aspect sequence charts give us the various combinations of signals, their aspect and indications conveyed to the driver of an approaching train. (Using light aspects)

Warner	**Outer**	**Home**	**Indication**
R	R	R	Stop at outer signal
R	G	G	Enter the station. Stop at starter of concerned line if 'ON'
G	G	G	Run through via main line all signals ahead are 'OFF'

Table 2.1 Approaching signals used in 2-aspect signalling

Distant	Home	Indication
Y	R	Stop at home signal
YY	Y	Enter on loop line. Stop at starter if ‘ON’.
G	Y	Enter on main line. Stop at starter.
G	G	Run through via main line

Table 2.2 Approaching signals used in MAUQ/MACL

Distant	Inner Distant	Home	Indication
YY	Y	R	Stop at home
YY	YY	Y	Enter on loop line. Stop at starter if ‘ON’
G	YY	Y	Enter on main line. Stop at starter.
G	G	G	Run through via main line

Table 2.3 Using two distant signals in approach (MACL)

Starter	Advanced Starter	Indication
R	R	Stand in rear of starter
G	R	Shunt upto advanced starter
G	G	Proceed line is clear

Table 2.4 Departure signals in 2-aspect signalling

Starter	Advanced Starter	Indication
R	R	Stand in rear of starter
Y	R	Shunt upto advanced starter
Y/G	G	Proceed line is clear

Table 2.5 Departure signals in M.A signalling

2.4 SUBSIDIARY SIGNALS

In the previous chapters we have seen the signals authorizing the drivers to enter the station from a block section by the use of reception signals; and

enter the block section from the station by the use of departure signals. These signals were, therefore, being used for reception and despatch of running trains. As per definition a “Running train” is a train which has started under an authority to proceed and has not completed its journey whereas “a train” is an engine with or without vehicles attached or self propelled vehicle with or without a trailer which cannot be readily lifted off the track. The signals, which control the movement of trains within the station section, are to be differentiated and convey different indication to the driver. These signals are (a) Shunt signals and (b) Calling on signals and are called “SUBSIDIARY SIGNALS”.

2.4.1 Shunt Signals

(a) Shunt signals authorize movement only at such slow speeds as to be able to stop short of any obstruction and control shunting movements.

(b) Shunt signals can be placed on a separate post by itself close to the ground or can be placed below a stop signal other than the first and last stop signal of a station.

(c) More than one shunt signal may be placed on the same post in which case the top-most signal shall apply to the extreme left hand line and the second shunt signal from the top shall apply to the next line from the left and so on.

(d) Shunt signal when taken ‘OFF’ authorities the driver to draw ahead with caution even though the stop signal, if any, above it is at ’ON’ position, and

(e) The shunt signal shall be either

(i) Disc type shunt signal.

(ii) Position light shunt signal.

(f) Under special instructions, a shunt signal may be a miniature arm.

(g) When a shunt signal is placed below a stop signal, it shall show no light in the "ON" position.

2.5 INTERLOCKING CONCEPTS

In order to ensure that the signalling system never provides unsafe (conflicting) signals and the points are not set for more than one train that might end up proceeding on to the same section of track and hence suffering a collision, various schemes have been developed to coordinate the settings of the points and the signals within the region controlled by a signalbox or signal cabin.

Mechanically operated interlocking: The most prevalent systems today (2003) are still mechanical interlocking schemes that coordinate the positions of the levers controlling the points with the signals governing that section of track and connected branches, loops, or sidings.

For instance, in one common scheme, a key that allows setting the points for a route has to be obtained from the block instrument, and as long as the key is removed the instrument cannot be set to provide Line Clear for a conflicting route. The wires that operate signals, and the rods that control points, are all interconnected in the lever frames at the signal cabins so that they are literally 'interlocked' - the position of one lever or key physically obstructs the movements of other levers and keys which control points or signals that can be set in conflicting ways.

Manually operated interlocking: This is a form of mechanical interlocking as well, but relies on the signalman to move about from one set of points and signals to another carrying with him the keys used to operate them. At small stations and on less busy branch lines various forms of manually operated mechanical interlocking are still [11/03] widespread. At points controlling catch sidings in hilly areas, often the interlocking is manual where the driver has to use a key provided by the stationmaster or signalman of the last station before the siding - the key is inserted into the interlock box which notifies the signal cabin and the points are then set for the main line and the signal is pulled off, giving the train authority to proceed. (This system is common in many hilly areas, although busier lines with catch sidings are being provided with automatically operating delayed signals where the points are controlled by a timer and are set to the main line only after the train has halted for the prescribed period of time.)

A common system in use was **Sequential Key Interlocking**, which saved on the installation of point rodding and instead relied on the signalman walking over with a key to lock or unlock points. As an example, consider a station with a main line and a loop line. To receive a train on the main line, a key is inserted into the signal frame in the cabin or platform, which allows the Outer and Home signals of the station to be pulled off.

In order to receive a train on the loop line instead, the key is used as before to pull off the Outer signal, but the Home is kept at danger. Instead, when the train has stopped at the Home signal the key is removed and taken to the facing points for the loop. The same key unlocks the points so they can be set for the loop; it also releases another key which has to be taken back and inserted in the signal frame at the platform to pull off the Home signal to let the train advance on to the loop.

The mechanism was such that only one of these two keys could be released at once; the second key did not allow the operation of the Outer signal, and it had to be taken back to the facing points of the loop in order to release the first key.

Electrically operated interlocking: In the more advanced electrical or electronic interlocking schemes, the points and signals are worked from one integrated mechanism in a signal cabin which features a display of the entire track layout with indications of sections that are occupied, free, set for reception or dispatch, etc. The interlocking is accomplished not by mechanical devices but by electrical circuitry - relays and switches in older electrical or electro pneumatic systems, and computerized circuits in the newer electronic systems.

Panel Interlocking(PI) is the system used in most medium-sized stations on IR. In this, the points and signals are worked by individual switches that control them. **Route Relay Interlocking**(RRI) is the system used in large and busy stations that have to handle high volumes of train movements. In this, an entire route through the station can be selected and all the associated points and signals along the route can be set at once by a switch for receiving, holding, blocking, or dispatching trains.

The description of the possible routes that can be set, and the corresponding dispositions of points and signals are found in the **locking table** and **selection table** for a station. The locking table lists the signals and points controlled; the levers at signal boxes (or control panels at control centres) which operate various signals and points; which signals and points are locked (and in what position) when other signals are pulled off or points set; which track circuits are clear or occupied; etc.

The requirement for having signalling system, different possible signalling systems with their underlying concepts and the indication of signals based on their location was discussed in this chapter. Also, the concepts of interlocking system and its types were discussed in detail.

CHAPTER 3

BASICS OF MODEL CHECKING

In the previous chapter we discussed about the various signalling concepts and the different techniques involved in interlocking system. This chapter deals about the importance and purpose of doing model checking before building any system either hardware or software after designing its prototype. There are also few examples which show the problems of not using model checking. The flow, architecture and process of model checking are also discussed.

Our reliance on the functioning of ICT systems (Information and Communication Technology) is growing rapidly. These systems are becoming more and more complex and are massively encroaching on daily life via the Internet and all kinds of embedded systems such as smart cards, hand-held computers, mobile phones, and high-end television sets. Services like electronic banking and teleshopping have become reality. The daily cash flow via the Internet is about 10^{12} million US dollars. Roughly 20% of the product development costs of modern transportation devices such as cars, high-speed trains, and airplanes is devoted to information processing systems.

ICT systems are universal and omnipresent. They control the stock exchange market, form the heart of telephone switches, are crucial to Internet technology, and are vital for several kinds of medical systems. Our reliance on embedded systems makes their reliable operation of large social importance. Besides offering a good performance in terms like response times and processing capacity, the absence of annoying errors is one of the major quality indications.

It is all about money. We are annoyed when our mobile phone malfunctions, or when our video recorder reacts unexpectedly and wrongly to our issued commands. These software and hardware errors do not threaten our lives, but may have substantial financial consequences for the manufacturer. Correct ICT systems are essential for the survival of a company. Dramatic examples are known. The bug in Intels Pentium II floating-point division unit in the early nineties caused a loss of about 475 million US dollars to replace faulty processors, and severely damaged Intels reputation as a reliable chip manufacturer. The software error in a baggage handling system postponed the opening of Denvers airport for 9 months, at a loss of 1.1 million US dollar per day. Twenty-four hours of failure of the worldwide online ticket reservation system of a large airplane company will cause its bankruptcy because of missed orders.

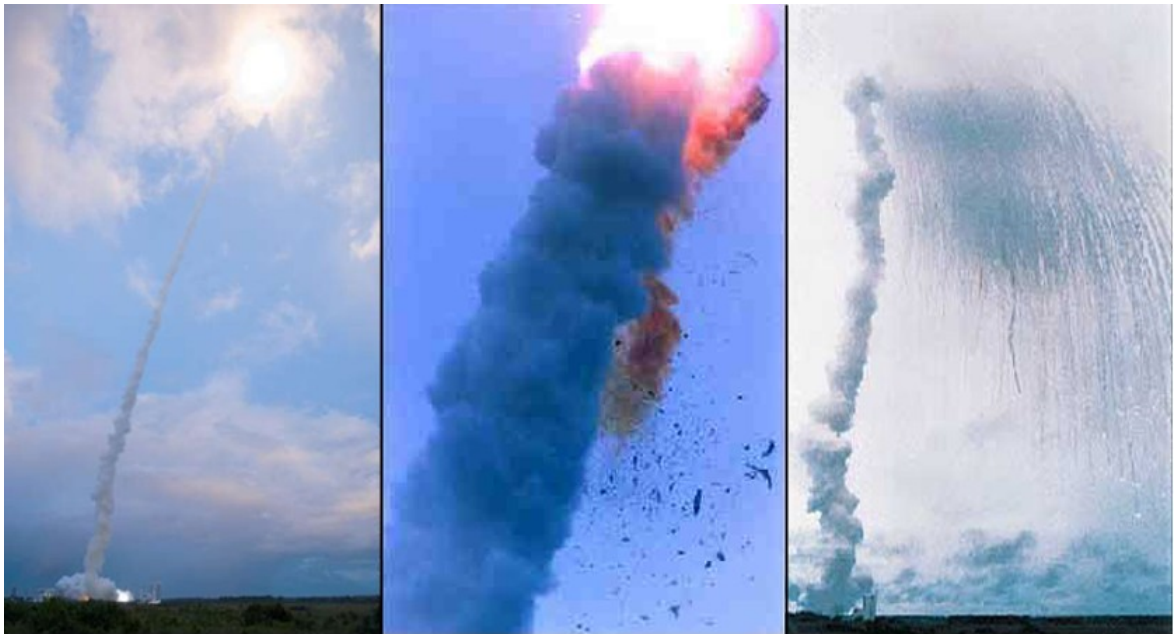


Figure 3.1 Ariane-5 explosion during launch

It is all about safety: errors can be catastrophic too. The fatal defects in the control software of the Ariane-5 missile (Figure 3.1), the Mars Pathfinder,

and the airplanes of the Airbus family led to headlines in newspapers all over the world and are notorious by now. Similar software is used for the process control of safety-critical systems such as chemical plants, nuclear power plants, traffic control and alert systems, and storm surge barriers. Clearly, bugs in such software can have disastrous consequences. For example, a software flaw in the control part of the radiation therapy machine Therac-25 caused the death of six cancer patients between 1985 and 1987 as they were exposed to an overdose of radiation.

The increasing reliance of critical applications on information processing leads us to state:

The reliability of ICT systems is a key issue in the system design process.

The magnitude of ICT systems, as well as their complexity, grows apace. ICT systems are no longer standalone, but are typically embedded in a larger context, connecting and interacting with several other components and systems. They thus become much more vulnerable to errors the number of defects grows exponentially with the number of interacting system components. In particular, phenomena such as concurrency and nondeterminism that are central to modeling interacting systems turn out to be very hard to handle with standard techniques. Their growing complexity, together with the pressure to drastically reduce system development time ("time-to-market"), makes the delivery of low-defect ICT systems an enormously challenging and complex activity.

Hardware and Software Verification

System verification techniques are being applied to the design of ICT systems in a more reliable way. Briefly, system verification is used to estab-

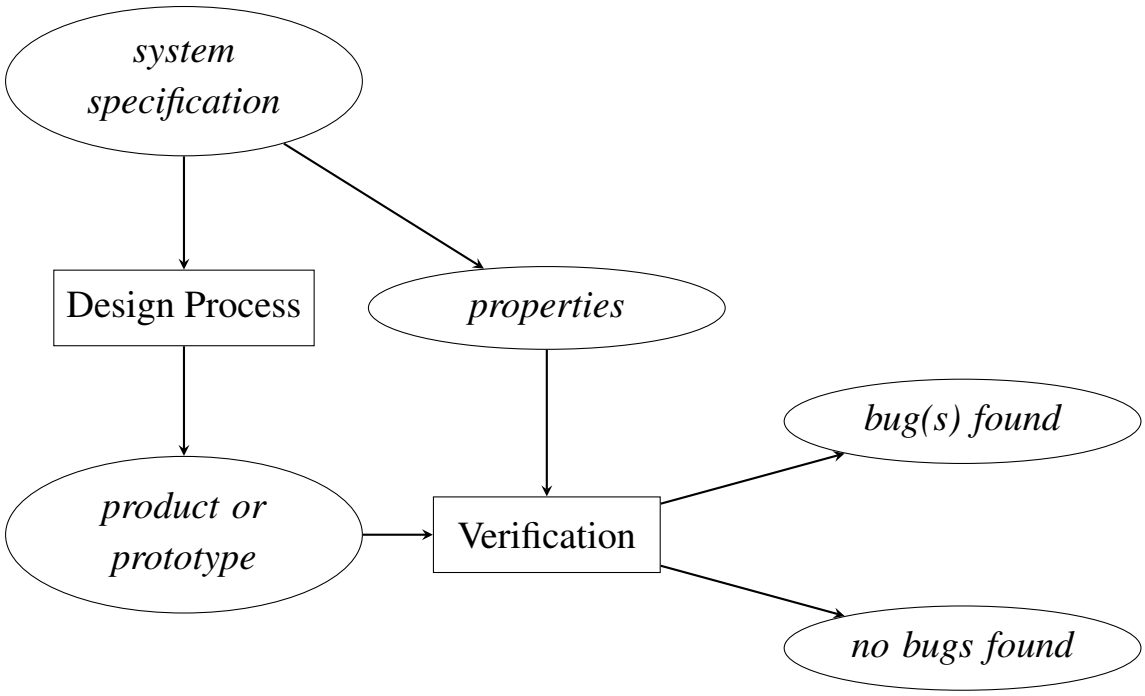


Figure 3.2 Schematic view of a posteriori system verification

lish that the design or product under consideration possesses certain properties. The properties to be validated can be quite elementary, e.g., a system should never be able to reach a situation in which no progress can be made (a deadlock scenario), and are mostly obtained from the systems specification. This specification prescribes what the system has to do and what not, and thus constitutes the basis for any verification activity. A defect is found once the system does not fulfill one of the specifications properties. The system is considered to be "correct" whenever it satisfies all properties obtained from its specification. So correctness is always relative to a specification, and is not an absolute property of a system. A schematic view of verification is depicted in Figure 3.2.

Software verification: Peer reviewing and testing are the major software verification techniques used in practice.

A *peer review* amounts to a software inspection carried out by a team of software engineers that preferably has not been involved in the development of the software under review. The uncompilable code is not executed, but analyzed completely statically. Empirical studies indicate that peer review provides an effective technique that catches between 31% and 93% of the defects with a median around 60%. While mostly applied in a rather ad hoc manner, more dedicated types of peer review procedures, e.g., those that are focused at specific error-detection goals, are even more effective. Despite its almost complete manual nature, peer review is thus a rather useful technique. It is therefore not surprising that some form of peer review is used in almost 80% of all software engineering projects. Due to its static nature, experience has shown that subtle errors such as concurrency and algorithm defects are hard to catch using peer review.

Software testing constitutes a significant part of any software engineering project. Between 30% and 50% of the total software project costs are devoted to testing. As opposed to peer review, which analyzes code statically without executing it, testing is a dynamic technique that actually runs the software. Testing takes the piece of software under consideration and provides its compiled code with inputs, called tests. Correctness is thus determined by forcing the software to traverse a set of execution paths, sequences of code statements representing a run of the software. Based on the observations during test execution, the actual output of the software is compared to the output as documented in the system specification. Although test generation and test execution can partly be automated, the comparison is usually performed by human beings. The main advantage of testing is that it can be applied to all sorts of software, ranging from application software (e.g., e-business software) to compilers and operating systems. As exhaustive testing

of all execution paths is practically infeasible; in practice only a small subset of these paths is treated. Testing can thus never be complete. That is to say, testing can only show the presence of errors, not their absence. Another problem with testing is to determine when to stop. Practically, it is hard, and mostly impossible, to indicate the intensity of testing to reach a certain defect density - the fraction of defects per number of uncommented code lines.

Studies have provided evidence that peer review and testing catch different classes of defects at different stages in the development cycle. They are therefore often used together. To increase the reliability of software, these software verification approaches are complemented with software process improvement techniques, structured design and specification methods (such as the Unified Modeling Language), and the use of version and configuration management control systems. Formal techniques are used, in one form or another, in about 10% to 15% of all software projects.

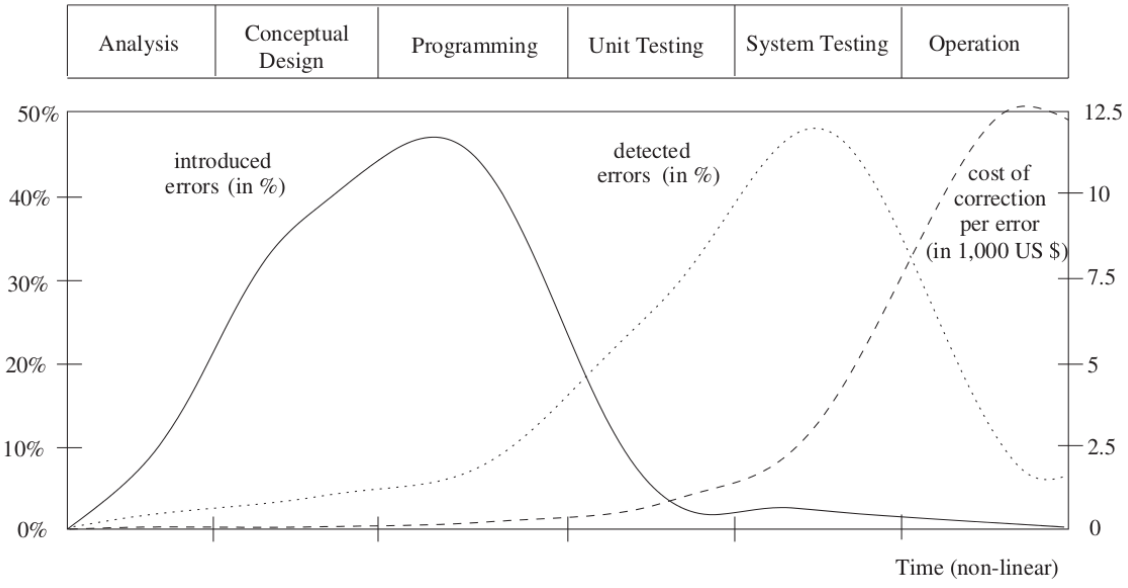


Figure 3.3 Software lifecycle and error introduction, detection, and repair costs

Catching software errors: the sooner the better. It is of great importance to locate software bugs. The slogan is: the sooner the better. The costs of repairing a software flaw during maintenance are roughly 500 times higher than a fix in an early design phase (see Figure 3.3). System verification should thus take place early stage in the design process.

About 50% of all defects are introduced during programming, the phase in which actual coding takes place. Whereas just 15% of all errors are detected in the initial design stages, most errors are found during testing. At the start of unit testing, which is oriented to discovering defects in the individual software modules that make up the system, a defect density of about 20 defects per 1000 lines of (uncommented) code is typical. This has been reduced to about 6 defects per 1000 code lines at the start of system testing, where a collection of such modules that constitutes a real product is tested. On launching a new software release, the typical accepted software defect density is about one defect per 1000 lines of code lines.

Errors are typically concentrated in a few software modules about half of the modules are defect free, and about 80% of the defects arise in a small fraction (about 20%) of the modules and often occur when interfacing modules. The repair of errors that are detected prior to testing can be done rather economically. The repair cost significantly increases from about $1000 (per error repair) in unit testing to a maximum of about $12,500 when the defect is demonstrated during system operation only. It is of vital importance to seek techniques that find defects as early as possible in the software design process: the costs to repair them are substantially lower, and their influence on the rest of the design is less substantial.

Hardware verification: Preventing errors in hardware design is vital. Hardware is subject to high fabrication costs; fixing defects after delivery to customers is difficult, and quality expectations are high. Whereas software defects can be repaired by providing users with patches or updates nowadays users even tend to anticipate and accept this hardware bug fixes after delivery to customers are very difficult and mostly require refabrication and redistribution. This has immense economic consequences. The replacement of the faulty Pentium II processors caused Intel a loss of about $475 million. Moore's law the number of logical gates in a circuit doubles every 18 months has proven to be true in practice and is a major obstacle to producing correct hardware. Empirical studies have indicated that more than 50% of all ASICs (Application-Specific Integrated Circuits) do not work properly after initial design and fabrication. It is not surprising that chip manufacturers invest a lot in getting their designs right. Hardware verification is a well-established part of the design process. The design effort in a typical hardware design amounts to only 27% of the total time spent on the chip; the rest is devoted to error detection and prevention.

Hardware verification techniques. Emulation, simulation, and structural analysis are the major techniques used in hardware verification.

Structural analysis comprises several specific techniques such as synthesis, timing analysis, and equivalence checking.

Emulation is a kind of testing. A reconfigurable generic hardware system (the emulator) is configured such that it behaves like the circuit under consideration and is then extensively tested. As with software testing, emulation amounts to providing a set of stimuli to the circuit and comparing the generated output with the expected output as laid down in the chip specification. To fully test the circuit, all possible input combinations in every

possible system state should be examined. This is impractical and the number of tests needs to be reduced significantly, yielding potential undiscovered errors.

With *simulation*, a model of the circuit at hand is constructed and simulated. Models are typically provided using hardware description languages such as Verilog or VHDL that are both standardized by IEEE. Based on stimuli, execution paths of the chip model are examined using a simulator. These stimuli may be provided by a user, or by automated means such as a random generator. A mismatch between the simulator's output and the output described in the specification determines the presence of errors. Simulation is like testing, but is applied to models. It suffers from the same limitations, though: the number of scenarios to be checked in a model to get full confidence goes beyond any reasonable subset of scenarios that can be examined in practice.

Simulation is the most popular hardware verification technique and is used in various design stages, e.g., at register-transfer level, gate and transistor level. Besides these error detection techniques, *hardware testing* is needed to find fabrication faults resulting from layout defects in the fabrication process.

3.1 MODEL CHECKING

In software and hardware design of complex systems, more time and effort are spent on verification than on construction. Techniques are sought to reduce and ease the verification efforts while increasing their coverage. Formal methods offer a large potential to obtain an early integration of verification in the design process, to provide more effective verification techniques, and to reduce the verification time.

Let us first briefly discuss the role of formal methods. To put it in

a nutshell, formal methods can be considered as "the applied mathematics for modeling and analyzing ICT systems". Their aim is to establish system correctness with mathematical rigor. Their great potential has led to an increasing use by engineers of formal methods for the verification of complex software and hardware systems. Besides, formal methods are one of the "highly recommended" verification techniques for software development of safety-critical systems according to, e.g., the best practices standard of the IEC (International Electrotechnical Commission) and standards of the ESA (European Space Agency). The resulting report of an investigation by the FAA (Federal Aviation Authority) and NASA (National Aeronautics and Space Administration) about the use of formal methods concludes that

Formal methods should be part of the education of every computer scientist and software engineer, just as the appropriate branch of applied maths is a necessary part of the education of all other engineers.

During the last two decades, research in formal methods has led to the development of some very promising verification techniques that facilitate the early detection of defects. These techniques are accompanied by powerful software tools that can be used to automate various verification steps. Investigations have shown that formal verification procedures would have revealed the exposed defects in, e.g., the Ariane-5 missile, Mars Pathfinder, Intels Pentium II processor, and the Therac-25 therapy radiation machine.

Model-based verification techniques are based on models describing the possible system behavior in a mathematically precise and unambiguous manner. It turns out that prior to any form of verification the accurate modeling of systems often leads to the discovery of incompleteness, ambiguities, and inconsistencies in informal system specifications. Such problems are usu-

ally only discovered at a much later stage of the design. The system models are accompanied by algorithms that systematically explore all states of the system model. This provides the basis for a whole range of verification techniques ranging from an exhaustive exploration (model checking) to experiments with a restrictive set of scenarios in the model (simulation), or in reality (testing). Due to unremitting improvements of underlying algorithms and data structures, together with the availability of faster computers and larger computer memories, model-based techniques that a decade ago only worked for very simple examples are nowadays applicable to realistic designs. As the starting point of these techniques is a model of the system under consideration, we have as a given fact that

Any verification using model-based techniques is only as good as the model of the system.

Model checking is a verification technique that explores all possible system states in a brute-force manner. Similar to a computer chess program that checks possible moves, a model checker, the software tool that performs the model checking, examines all possible system scenarios in a systematic manner. In this way, it can be shown that a given system model truly satisfies a certain property. It is a real challenge to examine the largest possible state spaces that can be treated with current means, i.e., processors and memories. State-of-the-art model checkers can handle state spaces of about 10^8 to 10^9 states with explicit state-space enumeration. Using clever algorithms and tailored data structures, larger state spaces (10^{20} up to even 10^{476} states) can be handled for specific problems. Even the subtle errors that remain undiscovered using emulation, testing and simulation can potentially be revealed using model checking.

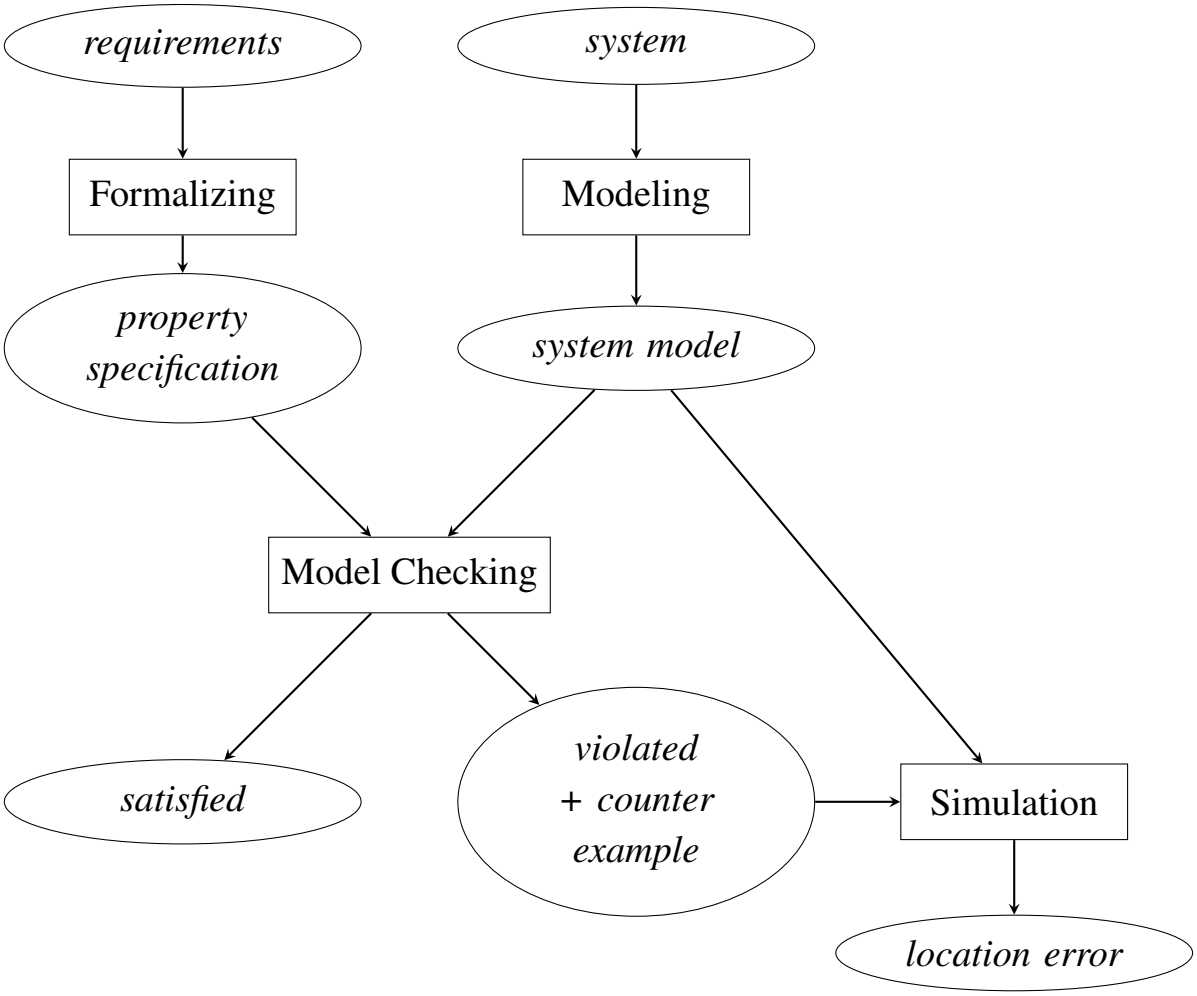


Figure 3.4 Schematic view of the model-checking approach

Typical properties that can be checked using model checking are of a qualitative nature: Is the generated result OK?, Can the system reach a deadlock situation, e.g., when two concurrent programs are waiting for each other and thus halting the entire system? But also timing properties can be checked: Can a deadlock occur within 1 hour after a system reset?, or, Is a response always received within 8 minutes? Model checking requires a precise and unambiguous statement of the properties to be examined. As with making an accurate system model, this step often leads to the discovery of several ambiguities and inconsistencies in the informal documentation. For instance,

the formalization of all system properties for a subset of the ISDN user part protocol revealed that 55%(!) of the original, informal system requirements were inconsistent.

The system model is usually automatically generated from a model description that is specified in some appropriate dialect of programming languages like C or Java or hardware description languages such as Verilog or VHDL. Note that the property specification prescribes what the system should do, and what it should not do, whereas the model description addresses how the system behaves. The model checker examines all relevant system states to check whether they satisfy the desired property. If a state is encountered that violates the property under consideration, the model checker provides a counterexample that indicates how the model could reach the undesired state. The counterexample describes an execution path that leads from the initial system state to a state that violates the property being verified. With the help of a simulator, the user can replay the violating scenario, in this way obtaining useful debugging information, and adapt the model (or the property) accordingly (see Figure 3.4).

Model checking has been successfully applied to several ICT systems and their applications. For instance, deadlocks have been detected in online airline reservation systems, modern e-commerce protocols have been verified, and several studies of international IEEE standards for in-house communication of domestic appliances have led to significant improvements of the system specifications. Five previously undiscovered errors were identified in an execution module of the Deep Space 1 spacecraft controller, in one case identifying a major design flaw. A bug identical to one discovered by model checking escaped testing and caused a deadlock during a flight experiment 96 million km from earth. In the Netherlands, model checking has

revealed several serious design flaws in the control software of a storm surge barrier that protects the main port of Rotterdam against flooding.

3.2 CHARACTERISTICS OF MODEL CHECKING

The principles of Model Checking are:

> *Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model.*

3.2.1 The Model-Checking Process

In applying model checking to a design the following different phases can be distinguished:

- Modeling phase:
  - model the system under consideration using the model description language of the model checker at hand;
  - as a first sanity check and quick assessment of the model perform some simulations;
  - formalize the property to be checked using the property specification language.
- Running phase: run the model checker to check the validity of the property in the system model.
- Analysis phase:
  - property satisfied? $\rightarrow$ check next property (if any);

- property violated? $\rightarrow$
  1. analyze generated counterexample by simulation;
  2. refine the model, design, or property;
  3. repeat the entire procedure.
- out of memory? $\rightarrow$ try to reduce the model and try again.

In addition to these steps, the entire verification should be planned, administered, and organized. This is called *verification organization*. We discuss these phases of model checking in somewhat more detail below.

Modeling The prerequisite inputs to model checking are a model of the system under consideration and a formal characterization of the property to be checked.

Models of systems describe the behavior of systems in an accurate and unambiguous way. They are mostly expressed using *finite-state automata*, consisting of a finite set of states and a set of transitions. States comprise information about the current values of variables, the previously executed statement (e.g., a program counter), and the like. Transitions describe how the system evolves from one state into another. For realistic systems, finite-state automata are described using a model description language such as an appropriate dialect/extension of C, Java, VHDL, or the like.

In order to improve the quality of the model, a simulation prior to the model checking can take place. Simulation can be used effectively to get rid of the simpler category of modeling errors. Eliminating these simpler errors before any form of thorough checking takes place may reduce the costly and time-consuming verification effort.

To make a rigorous verification possible, properties should be described in a precise and unambiguous manner. This is typically done using a property

specification language. We focus in particular on the use of a *temporal logic* as a property specification language, a form of modal logic that is appropriate to specify relevant properties of ICT systems. In terms of mathematical logic, one checks that the system description is a model of a temporal logic formula. This explains the term "model checking". Temporal logic is basically an extension of traditional propositional logic with operators that refer to the behavior of systems over time. It allows for the specification of a broad range of relevant system properties such as functional correctness (does the system do what it is supposed to do?), reachability (is it possible to end up in a deadlock state?), safety ("something bad never happens"), liveness ("something good will eventually happen"), fairness (does, under certain conditions, an event occur repeatedly?), and real-time properties (is the system acting in time?).

Although the aforementioned steps are often well understood, in practice it may be a serious problem to judge whether the formalized problem statement (model + properties) is an adequate description of the actual verification problem. This is also known as the validation problem. The complexity of the involved system, as well as the lack of precision of the informal specification of the systems functionality, may make it hard to answer this question satisfactorily. Verification and validation should not be confused. Verification amounts to check that the design satisfies the requirements that have been identified, i.e., verification is "check that we are building the thing right". In validation, it is checked whether the formal model is consistent with the informal conception of the design, i.e., validation is "check that we are verifying the right thing".

Running the Model Checker The model checker first has to be initialized by appropriately setting the various options and directives that may be used to carry out the exhaustive verification. Subsequently, the actual model checking takes place. This is basically a solely algorithmic approach in which the validity of the property under consideration is checked in all states of the system model.

Analyzing the Results There are basically three possible outcomes: the specified property is either valid in the given model or not, or the model turns out to be too large to fit within the physical limits of the computer memory.

In case the property is valid, the following property can be checked, or, in case all properties have been checked, the model is concluded to possess all desired properties.

Whenever a property is falsified, the negative result may have different causes. There may be a *modeling error*, i.e., upon studying the error it is discovered that the model does not reflect the design of the system. This implies a correction of the model, and verification has to be restarted with the improved model. This reverification includes the verification of those properties that were checked before on the erroneous model and whose verification may be invalidated by the model correction! If the error analysis shows that there is no undue discrepancy between the design and its model, then either a *design error* has been exposed, or a *property error* has taken place. In case of a design error, the verification is concluded with a negative result, and the design (together with its model) has to be improved. It may be the case that upon studying the exposed error it is discovered that the property does not reflect the informal requirement that had to be validated. This implies a modification of the property, and a new verification of the model has to be

carried out. As the model is not changed, no reverification of properties that were checked before has to take place. The design is verified if and only if all properties have been checked with respect to a valid model.

Whenever the model is too large to be handled - state spaces of real-life systems may be many orders of magnitude larger than what can be stored by currently available memories - there are various ways to proceed. A possibility is to apply techniques that try to exploit implicit regularities in the structure of the model. Examples of these techniques are the representation of state spaces using symbolic techniques such as binary decision diagrams or partial order reduction. Alternatively, rigorous abstractions of the complete system model are used. These abstractions should preserve the (non-)validity of the properties that need to be checked. Often, abstractions can be obtained that are sufficiently small with respect to a single property. In that case, different abstractions need to be made for the model at hand. Another way of dealing with state spaces that are too large is to give up the precision of the verification result. The probabilistic verification approaches explore only part of the state space while making a (often negligible) sacrifice in the verification coverage.

Verification Organization The entire model-checking process should be well organized, well structured, and well planned. Industrial applications of model checking have provided evidence that the use of version and configuration management is of particular relevance. During the verification process, for instance, different model descriptions are made describing different parts of the system, various versions of the verification models are available (e.g., due to abstraction), and plenty of verification parameters (e.g., model-checking options) and results (diagnostic traces, statistics) are available. This

information needs to be documented and maintained very carefully in order to manage a practical model-checking process and to allow the reproduction of the experiments that were carried out.

3.2.2 Strengths and Weaknesses

The strengths of model checking are:

- It is a general verification approach that is applicable to a wide range of applications such as embedded systems, software engineering, and hardware design.
- It supports *partial verification*, i.e., properties can be checked individually, thus allowing focus on the essential properties first. No complete requirement specification is needed.
- It is not vulnerable to the likelihood that an error is exposed; this contrasts with testing and simulation that are aimed at tracing the most probable defects.
- It provides *diagnostic information* in case a property is invalidated; this is very useful for debugging purposes.
- It is a potential "push-button" technology; the use of model checking requires neither a high degree of user interaction nor a high degree of expertise.
- It enjoys a rapidly increasing *interest by industry*; several hardware companies have started their in-house verification labs, job offers with required skills in model checking frequently appear, and commercial model checkers have become available.

- It can be easily *integrated* in existing development cycles; its learning curve is not very steep, and empirical studies indicate that it may lead to shorter development times.

- It has a *sound and mathematical underpinning*; it is based on theory of graph algorithms, data structures, and logic.

The weaknesses of model checking are:

- It is mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains.

- Its applicability is subject to *decidability issues*; for infinite-state systems, or reasoning about abstract data types (which requires undecidable or semi-decidable logics), model checking is in general not effectively computable.

- It verifies a *system model*, and not the actual system (product or prototype) itself; any obtained result is thus as good as the system model. Complementary techniques, such as testing, are needed to find fabrication faults (for hardware) or coding errors (for software).

- It checks only *stated requirements*, i.e., there is no guarantee of completeness. The validity of properties that are not checked cannot be judged.

- It suffers from the *state-space explosion* problem, i.e., the number of states needed to model the system accurately may easily exceed the amount of available computer memory. Despite the development of

several very effective methods to combat this problem, models of realistic systems may still be too large to fit in memory.

- Its usage requires some *expertise* in finding appropriate abstractions to obtain smaller system models and to state properties in the logical formalism used.

- It is not guaranteed to yield correct results: as with any tool, a model checker may contain software defects.

- It does not allow checking *generalizations*: in general, checking systems with an arbitrary number of components, or parameterized systems, cannot be treated. Model checking can, however, suggest results for arbitrary parameters that may be verified using proof assistants.

We believe that one can never achieve absolute guaranteed correctness for systems of realistic size. Despite the above limitations we conclude that

Model checking is an effective technique to expose potential design errors.

Through this chapter we discussed about the importance and purpose of doing model checking before building any system either hardware or software after designing its prototype. We also saw few examples which show the problems of not using model checking. The flow, architecture and process of model checking were also discussed.

CHAPTER 4

SURVEY OF RELATED WORKS

In the previous chapter we saw about the importance and different concepts in model checking. This chapter will inform you about the various related works done by researchers from around the world.

4.1 VERIFICATION IN UK RAILWAYS USING LADDER LOGIC

In the paper titled "Automated Verification of Signalling Principles in Railway Interlocking Systems" [3], the authors have used ladder logic to develop the software. Many experienced Engineers will try different use cases to test the software, which are listed in signalling books. The ladder logic is translated into propositional logic, to check the correctness of signalling system. The signalling system is represented in ladder logic and the safety conditions are represented in propositional logic. Thus, the verification of signalling system is done by verifying the propositional logic.

4.2 CSP TO MODEL AND FDR TO DO MODEL CHECKING

A paper titled "Model Checking Railway Interlocking Systems" [8], the author uses CSP to model the Railways interlocking system and FDR as model checking. The formal model of the Signalling Principles is called the *Principle Model*. The main aim is to translate the Control Tables into a minimal interlocking model, which is then model checked against the Principles Model.

4.3 INTERFACE TOOL TO CONVERT ASM TO SMV

In another paper titled "Model Checking Support for the ASM High-Level Language" [1], Abstract State Machines has been successfully converted to SMV, using mathematical formulas to verify. The contribution that the authors have made is developing an interface between ASM Workbench and SMV Model Checker. The ASM Workbench is a tool environment, which includes a type checker and a simulator for ASMs. SMV was chosen as a typical representative of a class of model checkers, based on transition systems, and could be easily replaced by any other model checker such as SVE or VIS.

4.4 MODEL CHECKING USING ASM

Another paper with the title "Modeling Large Railway Interlockings and Model Checking Small Ones" [9], the authors describe about the results to date of a feasibility study on model checking to be applied to railway interlockings. The target is a high level description of the interlocking systems, which is the logical view of its operation. Abstract State Machines(ASM) has been used to model the semantics of control tables. The reason to use ASM is that the resulting formal model is easier to read and understand. The formal model is transformed into NuSMV code by using a tool interface.

4.5 MODELING USING FSM AND CHECKING USING NUSMV

In an interesting paper titled "Automatic generation and verification of railway interlocking control tables using FSM and NuSMV" [4], the authors have detailed about the generation and verification of control tables in four steps.

1. Graphical Signalling Layout Planner

2. Route Table Generator

3. Control Table Generator

4. Control Table Verifier

This paper provides a complete tool for automating the generation of a safe Control Tables to be used for Railways.

4.6 MODEL CHECKING SAFETY-CRITICAL SYSTEMS

In the paper titled "Formal Methods in Development and Testing of Safety-Critical Systems: Railway Interlocking System" [2], the authors model a real-time system using formal methods in functional specification and perform verification for safety-critical systems. Safety-critical systems are often modeled by reactive systems. These reactive systems are used with applications that have high reliability and safety requirements. This leads to the rise of precise formal specification.

4.7 VERIFICATION USING COLORED PETRINETS

In another paper with the title "Verification of Railway Interlocking Tables using Colored Petri Nets"[7], the author models the interlocking tables using Colored Petri Nets(CPN). The CPN model comprises of two components namely *Signalling Layout Model* and *Interlocking Model*. ML functions are used on arc inscription in the Interlocking Model. These ML functions can be generated directly from the content of control table using Extensible Stylesheet Language Transformations$(XSLT)$, thus making CPN model to be easily reused and allowing other control tables to be built rapidly.

4.8 MODEL CHECKING USING BOOLEAN LOGIC

In a different paper titled “A logic approach to decision taking in a railway interlocking system using Maple”[5], the authors model the scenario using Boolean logic and independent from topology of stations. According to this model, any proposed situation is safe if and only if a certain set of formulae are consistent. The main procedure analyses the safety of a proposed situation and returns, if they exist, the sections where a collision could possibly take place. The reason for using Maple is that the code is brief for it.

4.9 DEVELOPING SAFETY-CRITICAL SYSTEMS USING V-MODEL

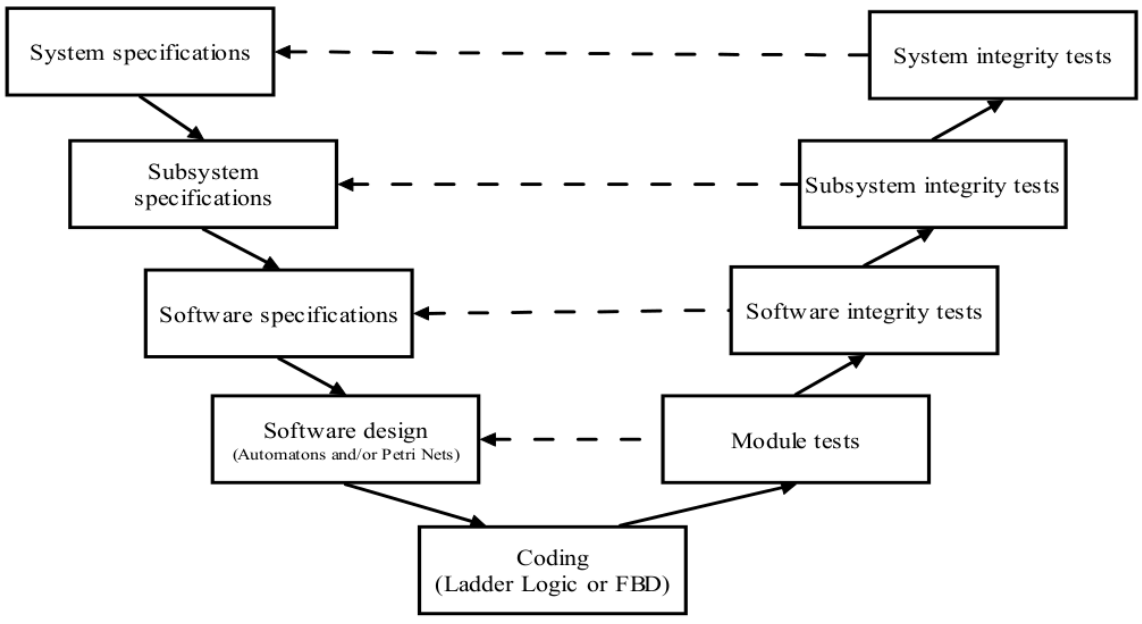


Figure 4.1 Interlocking development lifecycle - the V-Model

In yet another paper titled “The Application of Automation Theory to Railway Signalization Systems: The Case of Turkish National Railway

Signalization Project" [6], the authors use *V-Model*, [Figure 4.1] for the development of safety critical software. As per the model, the system specifications are determined. The sub-system and software specifications are derived from system specifications. Using well-defined software specifications, the software is designed using formal or semi-formal methods (such as Automatons and Petri Nets). Then the coding phase which happens, ladder diagrams or Functional Block Diagrams (FBD) are used.

This chapter gave us a bird's eye view of related works done by various researchers. These techniques give us a basic idea of how to build the solution for model checking Indian Railways.

CHAPTER 5

NuSMV

We saw about the various related works done by researchers across the globe in previous chapter. Basics of a famous model checking tool named NuSMV, will be discussed in this chapter. Different types of threading execution of the program will also be seen in detail.

NuSMV is a symbolic model checker developed as a joint project between:

- The Embedded Systems Unit in the Center for Information Technology at FBK-IRST.
- The Model Checking group at Carnegie Mellon University , the Mechanized Reasoning Group at University of Genova.
- The Mechanized Reasoning Group at University of Trento.

NuSMV is a reimplementation and extension of SMV, the first model checker based on BDDs. NuSMV has been designed to be an open architecture for model checking, which can be reliably used for the verification of industrial designs, as a core for custom verification tools, as a testbed for formal verification techniques, and applied to other research areas.

NuSMV2, combines BDD-based model checking component that exploits the CUDD library developed by Fabio Somenzi at Colorado University and SAT-based model checking component that includes an RBC-based Bounded Model Checker, which can be connected to the Minisat SAT Solver and/or to the ZChaff SAT Solver. The University of Genova has contributed

SIM, a state-of-the-art SAT solver used until version 2.5.0, and the RBC package use in the Bounded Model Checking algorithms.

The input language of NuSMV is designed to allow for the description of Finite State Machines (FSMs from now on) which range from completely synchronous to completely asynchronous, and from the detailed to the abstract. One can specify a system as a synchronous Mealy machine, or as an asynchronous network of nondeterministic processes. The language provides for modular hierarchical descriptions, and for the definition of reusable components. Since it is intended to describe finite state machines, the only data types in the language are finite ones - booleans, scalars and fixed arrays. Static data types can also be constructed.

The primary purpose of the NuSMV input is to describe the transition relation of the FSM; this relation describes the valid evolutions of the state of the FSM. In general, any propositional expression in the propositional calculus can be used to define the transition relation. This provides a great deal of flexibility, and at the same time a certain danger of inconsistency. For example, the presence of a logical contradiction can result in a deadlock - a state or states with no successor. This can make some specifications vacuously true, and makes the description unimplementable. While the model checking process can be used to check for deadlocks, it is best to avoid the problem when possible by using a restricted description style. The NuSMV system supports this by providing a parallel-assignment syntax. The semantics of assignment in NuSMV is similar to that of single assignment data flow language. By checking programs for multiple parallel assignments to the same variable, circular assignments, and type errors, the interpreter insures that a program using only the assignment mechanism is implementable. Consequently, this fragment of the language can be viewed as a description language, or a pro-

gramming language.

5.1 SYNCHRONOUS SYSTEMS

5.1.1 Single Process Example

Consider the following simple program in the NuSMV language:

```
MODULE main
 VAR
  request : boolean;
  state   : {ready, busy};
 ASSIGN
  init(state) := ready;
  next(state) := case
                  state = ready & request = TRUE : busy;
                  TRUE : {ready, busy};
                 esac;
```

The space of states of the FSM is determined by the declarations of the state variables (in the above example request and state). The variable request is declared to be of (predefined) type boolean. This means that it can assume the (boolean) values FALSE and TRUE. The variable state is a scalar variable, which can take the symbolic values ready or busy. The following assignment sets the initial value of the variable state to ready. The initial value of request is completely unspecified, i.e. it can be either FALSE or TRUE. The transition relation of the FSM is expressed by defining the value of variables in the next state (i.e. after each transition), given the value of variables in the current states (i.e. before the transition). The case segment sets the next value of the variable state to the value busy (after the colon) if its current value is ready and request is TRUE. Otherwise (the TRUE before the colon) the next value for state can be any in the set ready, busy. The variable request is not assigned. This means that there are no constraints on its values,

and thus it can assume any value. request is thus an unconstrained input to the system.

5.1.2 Binary Counter

The following program illustrates the definition of reusable modules and expressions. It is a model of a three bit binary counter circuit. The order of module definitions in the input file is not relevant.

```
MODULE counter_cell(carry_in)
 VAR
  value : boolean;
 ASSIGN
  init(value) := FALSE;
  next(value) := value xor carry_in;
 DEFINE
  carry_out := value & carry_in;
MODULE main
 VAR
  bit0 : counter_cell(TRUE);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);
```

The FSM is defined by instantiating three times the module type counter_cell in the module main, with the names bit0, bit1 and bit2 respectively. The counter_cell module has one formal parameter carry_in. In the instance bit0, this parameter is given the actual value TRUE. In the instance bit1, carry_in is given the value of the expression bit0.carry_out. This expression is evaluated in the context of the main module. However, an expression of the form '*a.b*' denotes component '*b*' of module '*a*', just as if the module '*a*' were a data structure in a standard programming language. Hence, the carry_in of module bit1 is the carry_out of module bit0.

The keyword 'DEFINE' is used to assign the expression value & carry_in to the symbol carry_out. A definition can be thought of as a variable with value (functionally) depending on the current values of other variables. The same effect could have been obtained as follows (notice that the *current* value of the variable is assigned, rather than the *next value*.):

```
VAR
 carry_out : boolean;
ASSIGN
 carry_out := value & carry_in;
```

Defined symbols do not require introducing a new variable, and hence do not increase the state space of the FSM. On the other hand, it is not possible to assign to a defined symbol a value non-deterministically. Another difference between defined symbols and variables is that while the type of variables is declared a priori, for definitions this is not the case.

5.2 ASYNCHRONOUS SYSTEMS

The previous examples describe synchronous systems, where the assignments statements are taken into account in parallel and simultaneously. NuSMV allows to model asynchronous systems. It is possible to define a collection of parallel processes, whose actions are interleaved, following an asynchronous model of concurrency. This is useful for describing communication protocols, or asynchronous circuits, or other systems whose actions are not synchronized (including synchronous circuits with more than one clock region).

5.2.1 Inverter Ring

The following program represents a ring of three asynchronous inverting gates.

```
MODULE inverter(input)
 VAR
  output : boolean;
 ASSIGN
  init(output) := FALSE;
  next(output) := !input;
MODULE main
 VAR
  gate1 : process inverter(gate3.output);
  gate2 : process inverter(gate1.output);
  gate3 : process inverter(gate2.output);
```

Among all the modules instantiated with the process keyword, one is non-deterministically chosen, and the assignment statements declared in that process are executed in parallel. It is implicit that if a given variable is not assigned by the process, then its value remains unchanged. Because the choice of the next process to execute is non-deterministic, this program models the ring of inverters independently of the speed of the gates.

We remark that the system is not forced to eventually choose a given process to execute. As a consequence the output of a given gate may remain constant, regardless of its input. In order to force a given process to execute infinitely often, we can use a fairness constraint. A fairness constraint restricts the attention of the model checker to only those execution paths along which a given formula is true infinitely often. Each process has a special variable called running which is TRUE if and only if that process is currently executing.

By adding the declaration:

```
FAIRNESS
 running
```

to the module inverter, we can effectively force every instance of inverter to execute infinitely often.

An alternative to using processes to model an asynchronous circuit is to allow all gates to execute simultaneously, but to allow each gate to choose non-deterministically to reevaluate its output or to keep the same output value. Such a model of the inverter ring would look like the following:

```
MODULE inverter(input)
 VAR
  output : boolean;
 ASSIGN
  init(output) := FALSE;
  next(output) := (!input) union output;
MODULE main
 VAR
  gate1 : inverter(gate3.output);
  gate2 : inverter(gate1.output);
  gate3 : inverter(gate2.output);
```

The union operator (set union) coerces its arguments to singleton sets as necessary. Thus, the next output of each gate can be either its current output, or the negation of its current input - each gate can choose non-deterministically whether to delay or not. As a result, the number of possible transitions from a given state can be as 2^n, where *n is the number of gates. This sometimes (but not always) makes it more expensive to represent the transition relation. We remark that in this case we cannot force the inverters to be effectively active infinitely often using a fairness declaration. In fact, a valid scenario for the synchronous model is the one where all the inverters are idle and assign to the next output the current value of output.*

5.2.2 Mutual Exclusion

The following program is another example of asynchronous model. It uses a variable semaphore to implement mutual exclusion between two asynchronous processes. Each process has four states: idle, entering, critical and exiting. The entering state indicates that the process wants to enter its critical region. If the variable semaphore is FALSE, it goes to the critical state, and sets semaphore to TRUE. On exiting its critical region, the process sets semaphore to FALSE again.

```
MODULE main
 VAR
  semaphore : boolean;
  proc1 : process user(semaphore);
  proc2 : process user(semaphore);
 ASSIGN
  init(semaphore) := FALSE;
MODULE user(semaphore)
 VAR
  state : {idle, entering, critical, exiting};
 ASSIGN
  init(state) := idle;
  next(state) :=
   case
    state = idle                  : {idle, entering};
    state = entering & !semaphore : critical;
    state = critical              : {critical, exiting};
    state = exiting               : idle;
    TRUE                          : state;
   esac;
  next(semaphore) :=
   case
    state = entering : TRUE;
    state = exiting  : FALSE;
```

```
    TRUE              : semaphore;
   esac;
FAIRNESS
 running
```

The basics of NuSMV, was discussed in this chapter in detail. Different types of threading execution of the program was also shown in detail.

CHAPTER 6

IMPLEMENTATION AND CASE STUDY

In previous chapter, we saw about basics and thread execution types in NuSMV. This chapter will say about the implementation details to generate verified control table entries for Indian Railways interlocking system.

6.1 INTRODUCTION

In implementing a solution to this problem, understanding the signalling system of Indian Railways requires much importance. The railway section or railway station is represented on a layout. The layout consists of track segments, signals and points. The track segments are segments of track line. A train moves on a track by occupying and releasing a track segment. Signals provide the visually encoded information for the train drivers. They specify information like the speed at which the train must move, track on which the train has been allocated to move. Points are the intersection points of two track segments that help a train to change between track lines. A point can be set as *normal* or *reverse*.

Every track segment has an unique ID in a given layout. They may also have one or more labels to be used for specifying route through the railway section. Signals have ID to denote them uniquely. Points also have unique ID to be specified. A route is a sequence of track segments starting from a signal. It also comprises of details about points concerned with the route, to be whether normal or reverse.

6.2 THE RAILWAY SECTION LAYOUT

An example railway section layout has been given in Figure A.1. In this example, there are two track lines with 17 track segments, 4 points, 14 signals and 9 labels. To give this layout as input, it must be specified using a formal model such as graph. All the labels and ID are specified as attibutes of vertices and edges in the graph.

This layout consists of 5 different types of signals namely, Calling on home, Home, Shunt, Starter and Advanced starter. The 14 signals can be classified into these types. 1B and 32B are calling on home. 1A and 32A are home. 9 and 17 are shunt. 3, 2, 6, 30, 31 and 27 are starter. 8 and 25 are advanced starter. The starter signals can be further classified into main line starter(2 and 31) and loop line starter(3, 6, 30 and 27). A train can move either towards up or towards down through a railway section. Hence it is important to provide signalling for both the directions.

The points 50, 52, 63 and 65 allow a train to move from main line to loop line or vice versa. A main line is the sequence of track segments that can allow a train to move from one railway section to another. A loop line is a sequence of track segments that branches from main line. When calling on home and shunt signal are given to the train driver, he need not stop the train at the specified track segment. The difference between calling on home and shunt signal is that the train moving on calling on home signal will have higher speed than shunt signal. The train following shunt signal will be moving at a very low speed of 10-20 kmph. Also, for train following shunt signal, there will be a shunter person to monitor the train movement which is not in calling on home.

6.3 FORMAL REPRESENTATION OF THE LAYOUT

The above layout is represented as a bidirectional graph to make it easier for a program to parse and identify the various aspects of a railway section. The track segments are represented as vertices using circular nodes. An edge connects two vertices if a train can move from one track segment to another. Labels for track segments are represented using elliptical nodes. Signals at a track segment are represented using rectangular nodes. Point IDs for specific track segments are represented using triangular nodes. The graph for above layout is given in Figure A.2.

6.4 REPRESENTATION OF LAYOUT GRAPH AS A FILE

The graph as shown in Figure A.2 is represented as text in a file and given as input to the program. The program parses the input file and recognizes the graph as adjacency list. Other information such as signals, point IDs and track labels are also parsed and recognized. The input file consists of 6 sections, each separated by the keyword 'END'.

The first section consists of neighbouring vertices where the first vertex on left denotes a track segment on the up side of the railway section with respect to the other track segment. The second section consists of the impossible path that a train can take at a point. Also, the point ID of a point is given with this info. The third section consists of signal information and the track segment at which it is positioned. The fourth section consists of labels given for track segments. If a label corresponds to platform at which a train must stop, it is given with an extra information '-P'. The fifth section consists of track info of track segments to which they belong to. The sixth section consists of signal and labels for which the control table needs to be generated. These denote the route end points as the control table will be generated. This

input format is given below. The program which generates the control table for a given layout is built using Ruby v1.9.3 programming language on Ubuntu 14.04 LTS OS.

6.5 FUNCTIONS OF MODULES

The input file will be parsed by Interface module to identify the graph as instances of every item in the layout such as, signals and labels. These objects will be used by other modules to generate verified control table entries. The input file name is specified in the file **input.rly** found in the main folder.

6.5.1 Routes Generator module

The Routes Generator module, makes use of a slightly modified version of Depth First Search Algorithm to come up with routes. In Home signal, there can be two types starting from the signal specified. In first type, the train passes through a railway section and stops at track where there is a label next to the specified label. In second type, the train stops at the next track of the track which has the label specified. In Calling on home signal, the train is operated by the driver manually without any tracks being controlled. In Starter signal, the train starts from the signal and moves till the track which has the label specified. In Advanced starter signal, the train starts from the signal and moves till the track before track which has the label specified. In Shunt signal, the train starts from the signal and moves to the track before track which has the label specified.

6.5.2 Control Table Generator module

In Control Table Generator module, the generated routes are analyzed to find out what must be the status of points for a train to follow the route. When there is a path that covers two track segments with same point ID, then

the point is in reverse status. In other case, when there is a path that covers one track segment with a point ID but the consecutive track segment does not has the same point ID, then the point is in normal status. For all the points in the given layout, the track segments that contain the point ID are listed in the field of paths for different routes.

6.5.3 Consistent Routes Combination Verifier module

The Consistent Routes Combination Verifier module, makes use of NuSMV model checker to verify if two trains following two different routes, will have a collision or not. A basic check must be made to ensure that no two routes should have different status of point which is common between them. For example, routes 1A-A and 1A-A1 cannot be enabled at same time, as the point 63 is reverse for 1A-A and normal for 1A-A1. This is called **points conflict**.

As the trains following calling on home and shunt signals differ from the trains following other signals, there is a need to come up with two different types of models. As an example, lets build models for 1A-A and 9-A, as shown below in Figures C.1 and C.2. The point 63 is set to reverse by 1A-A and so the train following 9-A can move till the end. If another route, for example 32A-C1 is considered, it requires the points 52 and 65 to be in normal and reverse respectively, making the train following 9-A route, to stop at 63AT.

This model is represented in NuSMV program as given in appendix section. The routes are in the form of modules. They are allowed to move as synchronous non-deterministic transition model from the main module.

Safety condition

The condition to be checked for non collision to ensure safety is given by the below LTL formula, where *train_1* is the train following a route and *train_2* is the train following a different route.

G !(train_1.track_id = train_2.track_id)

Reason for conflict between two routes

The reason for two routes to be conflicting or not, is given below for every two combinations of signal types. There are five signal types and distinct combinations are fifteen. For every combination, the direction of train travel can be in same direction or opposite direction. This gives a total of thirty combinations.

1. *Home - Home:*
    1.1 *Same direction:* The routes cannot conflict as points conflict.
    1.2 *Opposite direction:* If the routes have any track segments in common, then they cause collision and hence they become conflicting routes.
2. *Home - Calling on home:*
    2.1 *Same direction:* If the routes have any track segments in common, then they cause collision and hence they become conflicting routes.
    2.2 *Opposite direction:* If the routes have any track segments in common or if the train following calling on home signal moves forward

till there is a point of isolation, then they cause collision and hence they become conflicting routes.

3. *Home - Starter:*

   3.1 *Same direction:* There can be no collisions as the track segments are already locked by either home or starter signal. So there can be no conflicting routes in this combination.

   3.2 *Opposite direction:* If the routes have any track segments in common, then they cause collision and hence they become conflicting routes.

4. *Home - Advanced starter:*

   4.1 *Same direction:* As there are no track segments in common, collision cannot happen. Hence there can be no conflicting routes in this combination.

   4.2 *Opposite direction:* If the routes have any track segments in common, then they cause collision and hence they become conflicting routes.

5. *Home - Shunt:*

   5.1 *Same direction:* If the routes have any track segments in common, then they cause collision and hence they become conflicting routes.

   5.2 *Opposite direction:* If the routes have any track segments in common or if the train following shunt signal moves forward till there is a point of isolation, then they cause collision and hence they become conflicting routes.

6. *Calling on home - Calling on home:*

   6.1 *Same direction:* As all the routes combination end up in points conflict, there can be no collision. Hence all the routes combination are not conflicting.

   6.2 *Opposite direction:* As the trains can proceed further ahead of the given path, till point of isolation is reached, it causes collision and become conflicting routes.

7. *Calling on home - Starter:*

   7.1 *Same direction:* As the trains can proceed further ahead of the given path, till point of isolation is reached, it causes collision and become conflicting routes.

   7.2 *Opposite direction:* If the routes have any track segments in common, then they cause collision and hence they become conflicting routes.

8. *Calling on home - Advanced starter:*

   8.1 *Same direction:* As there is a huge gap for a train following advanced starter and calling on home, there cannot be collision between trains. Hence there are no conflicting routes in this combination.

   8.2 *Opposite direction:* If the routes have any track segments in common, then they cause collision and hence they become conflicting routes.

9. *Calling on home - Shunt:*

9.1 *Same direction:* If the routes have any track segments in common, then they cause collision and hence they become conflicting routes.

9.2 *Opposite direction:* If the train moves forward till there is a point of isolation, then they cause collision and hence they become conflicting routes.

10. *Starter - Starter:*

10.1 *Same direction:* As all the route combinations have points conflict, there are no conflicting routes in this combination.

10.2 *Opposite direction:* As there are no track segments in common, there is no collision between trains. Hence there are no conflicting routes in this combination.

11. *Starter - Advanced starter:*

11.1 *Same direction:* As there are no track segments in common, there are no conflicting routes in this combination.

11.2 *Opposite direction:* As there are no track segments in common, there are no conflicting routes in this combination.

12. *Starter - Shunt:*

12.1 *Same direction:* If the routes have any track segments in common or if the train following shunt signal moves forward till there is a point of isolation, then they cause collision and hence they become conflicting routes.

12.2 *Opposite direction:* If the routes have any track segments in common, then they cause collision and hence they become conflicting routes.

13. *Advanced starter - Advanced starter:*

    13.1 *Same direction:* There cannot be any combinations between same routes, there are no conflicting routes in this combination.

    13.2 *Opposite direction:* As there are no track segments in common, there are no conflicting routes in this combination.

14. *Advanced starter - Shunt:*

    14.1 *Same direction:* As there is a huge gap between the trains following advanced starter & shunt signals and the train following shunt signal is being controlled by a shunter & operates at very low speed, there cannot be collision. Hence there are no conflicting routes in this combination.

    14.2 *Opposite direction:* As the train following shunt signal must not make the driver confused with advanced starter signal too, these combinations are not allowed. Hence all these route combinations are conflicting.

15. *Shunt - Shunt:*

    15.1 *Same direction:* As the route combinations have point conflict, there can be no conflicting routes.

    15.2 *Opposite direction:* As the trains following shunt signals are operated at low speed and controlled by shunter, only those routes

which end at track segments with same label, cause collision. Hence they are only conflicting routes.

The final result is presented in a PDF as tabular format generated using LaTeXsoftware.

6.6 PERFORMANCE EVALUATION

Performance of this system is a measure of likeliness of the already generated control table entries being generated. On comparing with the sample data, it is found that the system is able to give 100% performance.

We saw how generation of verified control table entries was done using Ruby programming language and NuSMV model checker. The functionality of every module in the system was also explained clearly.

CHAPTER 7

CONCLUSION AND FUTURE WORK

This work provides us the control table which is verified using model checking technique(LTL). The choice of Ruby as programming language has helped to execute terminal commands to run NuSMV programs and compile the LaTeXfile inside the program. Also, the programming language was easy to build and trace for bugs. The least required version of Ruby to run this is 1.9.3. The website for this project can be found in the link: *http://modelchecking.github.io/identify_conflicting_routes*.

As future work, NuSMV can be used to model an actual railway station working with the generated control table as input and verify if the generated control table has any flaws or completely safe. Also, the signalling principles can be specified using a formal language which can be further utilized to check properties on its working mechanism.

APPENDIX A

RAILWAY SECTION LAYOUT AND GRAPH

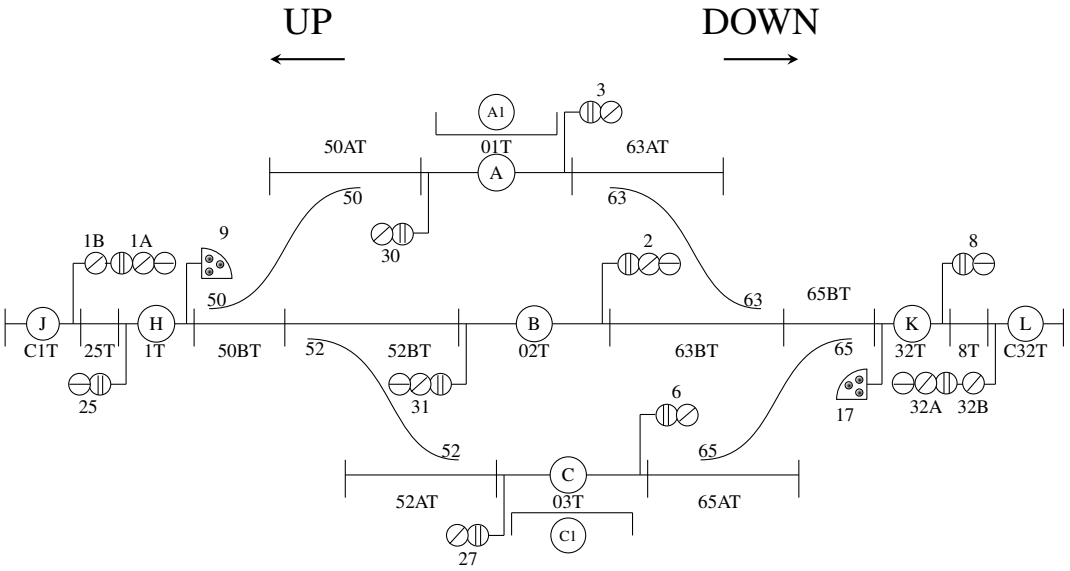


Figure A.1 Sample layout of a railway section

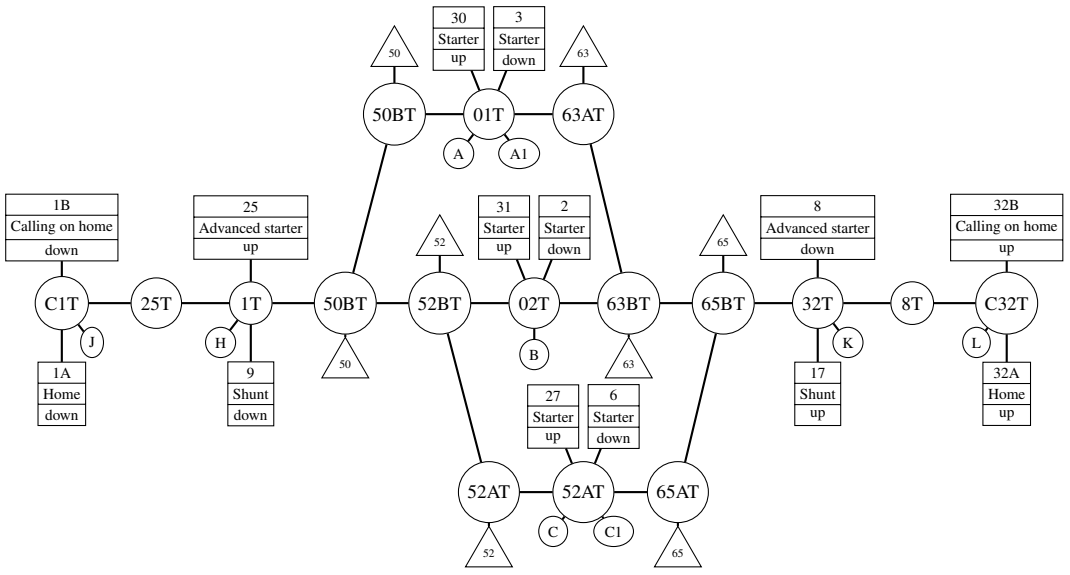


Figure A.2 Graph representation of sample railway section layout

APPENDIX B

INPUT

```
C1T 25T
25T 1T
1T 50BT
50BT 50AT
50BT 52BT
52BT 52AT
52BT 02T
02T 63BT
50AT 01T
01T 63AT
63AT 63BT
63BT 65BT
52AT 03T
03T 65AT
65AT 65BT
65BT 32T
32T 8T
8T C32T
END
52BT 50BT 50AT 50
02T 52BT 52AT 52
02T 63BT 63AT 63
63BT 65BT 65AT 65
END
C1T (1B down calling_on_home)
C1T (1A down home)
1T (25 up advanced_starter)
1T (9 down shunt)
01T (30 up starter)
02T (31 up starter)
```

```
03T (27 up starter)
01T (3 down starter)
02T (2 down starter)
03T (6 down starter)
32T (17 up shunt)
32T (8 down advanced_starter)
C32T (32A up home)
C32T (32B up calling_on_home)
END
C1T J
1T H
01T A A1-P
02T B
03T C C1-P
32T K
C32T L
END
T1 C1T 25T 1T 50BT 52BT 02T 63BT 65BT 32T 8T C32T
T2 50AT 01T 63AT
T3 52AT 03T 65AT
END
1A A A1 B C C1
1B A B C
2 K
3 K
6 K
8 L
9 A B C
17 A B C
25 J
27 H
30 H
31 H
32A A A1 B C C1
```

```
65  32B  A  B  C
```

code/input/sample_layout.txt

APPENDIX C

MODEL OF ROUTE

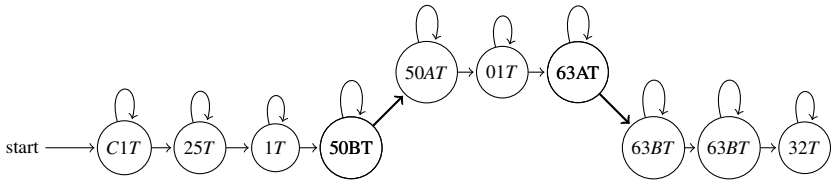


Figure C.1 Model of route 1A-A through the Railway Section

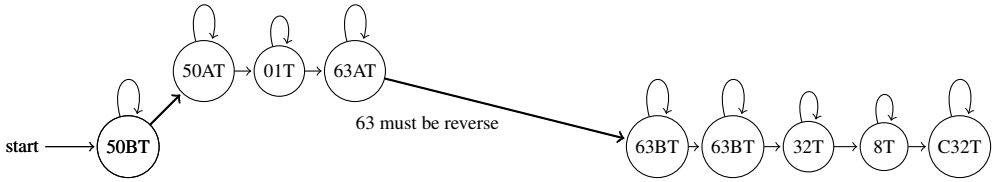


Figure C.2 Model of route 9-A through the Railway Section

APPENDIX D

NuSMV PROGRAM FOR MODEL

```
1  -- *****
2  -- EVERY ROUTE IS A MODULE. A TRAIN WILL MOVE IN A ROUTE.
3  -- *****
4
5  MODULE route_1A_A()
6    VAR
7      track_id : {"25T", "1T", "50BT", "50AT", "01T", "63AT", "63
       BT", "65BT", "32T"};
8    ASSIGN
9      init(track_id) := "25T";
10     next(track_id) := case
11                         track_id = "25T" : {"25T", "1T"};
12                         track_id = "1T" : {"1T", "50BT"};
13                         track_id = "50BT" : {"50BT", "50AT"};
14                         track_id = "50AT" : {"50AT", "01T"};
15                         track_id = "01T" : {"01T", "63AT"};
16                         track_id = "63AT" : {"63AT", "63BT"};
17                         track_id = "63BT" : {"63BT", "65BT"};
18                         track_id = "65BT" : {"65BT", "32T"};
19                         TRUE : track_id;
20                       esac;
21
22 MODULE route_9_A(point_63, point_65)
23   VAR
24     track_id : {"50BT", "50AT", "01T", "63AT", "63BT", "65BT",
       "32T", "8T", "C32T"};
25   ASSIGN
26     init(track_id) := "50BT";
27     next(track_id) := case
28                         track_id = "50BT" : {"50BT", "50AT"};
```

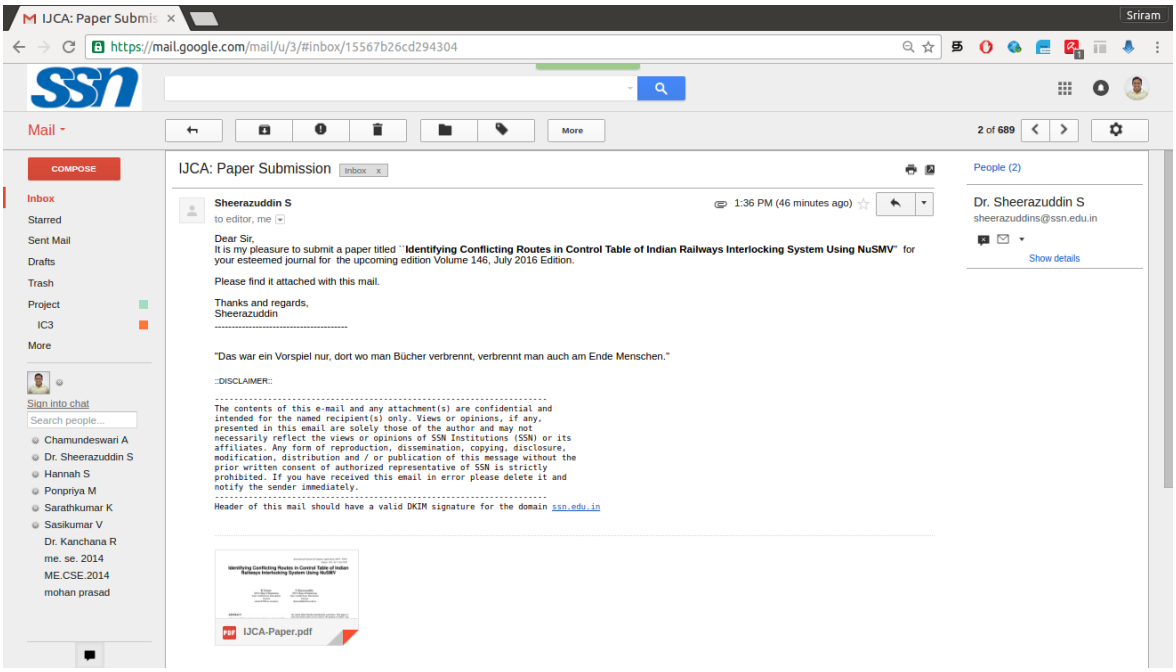
```
29                          track_id = "50AT" : {"50AT", "01T"};
30                          track_id = "01T" : {"01T", "63AT"};
31                          track_id = "63AT" & !point_63 : {"63AT"
    , "63BT"};
32                          track_id = "63BT" : {"63BT", "65BT"};
33                          track_id = "65BT" : {"65BT", "32T"};
34                          track_id = "32T" : {"32T", "8T"};
35                          track_id = "8T" : {"8T", "C32T"};
36                          TRUE : track_id;
37                        esac;
38
39  MODULE main()
40    VAR
41      point_63 : boolean;
42      point_65 : boolean;
43      train_1A_A : route_1A_A();
44      train_9_A : route_9_A(point_63, point_65);
45    ASSIGN
46      init(point_63) := FALSE;
47      init(point_65) := TRUE;
48      next(point_63) := point_63;
49      next(point_65) := point_65;
50
51  LTLSPEC
52    G !(train_1A_A.track_id = train_9_A.track_id);
```

code/output/sample_layout_txt/NuSMV/model_1A–A_9–A.smv

APPENDIX E

JOURNAL DETAILS

A paper titled “Identifying Conflicting Routes in Control Table of Indian Railways Interlocking System Using NuSMV” was submitted to the International Journal of Computer Applications(0975 - 8887) Volume 146 - No.*, July 2016.



REFERENCES

1. Giuseppe Del Castillo and Kirsten Winter. Model checking support for the ASM high-level language. In *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*, pages 331–346, 2000.

2. Tomas Hlavaty, Libor Preucil, and Petr Stepan. Case study: Formal specification and verification of railway interlocking system. In *27th EUROMICRO Conference 2001: A Net Odyssey, 4-6 September 2001, Warsaw, Poland*, pages 258–263, 2001.

3. Karim Kanso, Faron Moller, and Anton Setzer. Automated verification of signalling principles in railway interlocking systems. *Electr. Notes Theor. Comput. Sci.*, 250(2):19–31, 2009.

4. Ahmad Mirabadi and Mohammad Bemani Yazdi. Automatic generation and verification of railway interlocking control tables using fsm and nusmv. In *Engineering Modelling 21 (2008) 1-4*, pages 57–63, 2008.

5. Eugenio Roanes-Lozano, Antonio Hernando, José-Antonio Alonso, and Luis M. Laita. A logic approach to decision taking in a railway interlocking system using maple. *Mathematics and Computers in Simulation*, 82 (1):15–28, 2011.

6. Mehmet T. Sylemez, Mustafa S. Durmu, Uur Yldrm, Serhat Trk, and Arcan Sonat. The application of automation theory to railway signalization

systems: The case of turkish national railway signalization project. In *18th IFAC World Congress, 2011, Italy*, pages 10752–10757, 2011.

7. Somsak Vanit-Anunchai. Experience using coloured petri nets to model railway interlocking tables. In *Proceedings 2nd French Singaporean Workshop on Formal Methods and Applications, FSFMA 2014, Singapore, 13th May 2014.*, pages 17–28, 2014.

8. K. Winter. Model checking railway interlocking systems. In *Computer Science 2002, Twenty-Fifth Australasian Computer Science Conference (ACSC2002), Monash University, Melbourne, Victoria, January/February 2002*, pages 303–310, 2002.

9. Kirsten Winter and Neil J. Robinson. Modelling large railway interlockings and model checking small ones. In *Computer Science 2003, Twenty-Sixth Australasian Computer Science Conference (ACSC2003), Adelaide, South Australia, February 2003*, pages 309–316, 2003.