# "Verification of railway interlocking systems and optimisation of railway traffic"

Cappart, Quentin

**Abstract**

Since the dawn of the nineteenth century, development of railway systems has taken a huge importance in many countries. Over the years, the number of trains, the number of tracks, the complexity of networks increase and are still increasing. Directing trains on efficient routes, stopping and cancelling them are some actions that railway operators must take in their everyday life in order to regulate the traffic. However, with its continual growth, the consequences of such actions become rapidly hard to predict. Bad decisions can lead to disastrous situations such as accidents or, in the best cases, to unnecessary delays leading to financial losses. Decisions and actions that could be taken manually in the past are now hard combinatorial problems that require computer based methods for their solving. In this context, the need of a reliable and efficient railway traffic management is crucial. Like any transportation system, three aspects must be considered: safety, availability and flu...

Document type : *Thèse (Dissertation)*

# Verification of Railway Interlocking Systems and Optimisation of Railway Traffic

Quentin Cappart

December 2017

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Applied Science in Engineering

Institute of Information and Communication Technologies,
Electronics and Applied Mathematics (ICTEAM)
Louvain School of Engineering (EPL)
Université catholique de Louvain (UCL)
Louvain-la-Neuve
Belgium

**Examining board**

| | |
|---|---|
| Prof. Pierre **Schaus**, *Supervisor* | UCL/ICTEAM, Belgium |
| Prof. Peter **Van Roy**, *President* | UCL/ICTEAM, Belgium |
| Prof. Yves **Deville** | UCL/ICTEAM, Belgium |
| Prof. Charles **Pecheur** | UCL/ICTEAM, Belgium |
| Prof. Axel **Legay** | INRIA, France |
| Dr. Luis-Fernando **Mejia** | Alstom, France |
| Dr. Renaud **De Landtsheer** | CETIC, Belgium |

# Abstract

Since the dawn of the nineteenth century, development of railway systems has taken a huge importance in many countries. Over the years, the number of trains, the number of tracks, the complexity of networks increase and are still increasing. Directing trains on efficient routes, stopping and cancelling them are some actions that railway operators must take in their everyday life in order to regulate the traffic. However, with its continual growth, the consequences of such actions become rapidly hard to predict. Bad decisions can lead to disastrous situations such as accidents or, in the best cases, to unnecessary delays leading to financial losses. Decisions and actions that could be taken manually in the past are now hard combinatorial problems that require computer based methods for their solving. In this context, the need of a reliable and efficient railway traffic management is crucial. Like any transportation system, three aspects must be considered: safety, availability and fluidity. Safety and availability belong to verification engineering while fluidity is related to optimisation.

A plethora of research on this field already exist. However, most of it suffers of a lack of scalability. They can only be used for small or medium stations. This thesis presents innovative approaches for tackling this problem. For each aspect, we propose a method, that is feasible in practice for stations of any size. Concretely, verification of safety is performed with a dedicated algorithm while availability is verified with Statistical Model Checking. Fluidity optimisation is carried out with Constraint Programming. The performance of these methods are analysed through three stations of the Belgian railway network.

# Acknowledgements

Even if there is only one author, achieving this thesis could not have been possible without the help, the support and the comments I received from many people. My first thanks go to my supervisor, Pierre Schaus, for its guidance all along this thesis. Finding a good match is not always easy but working with Pierre was for me a pleasure. Pierre, thank you for providing me with the freedom I needed, for always being there when required and for continuously giving me new challenges.

During my thesis, I had the great opportunity to collaborate with many people from different backgrounds. I thank Christophe Ponsard, Jean-Jacques Gehrenbeck, Yoann Guyot, Raphaël Michel, Michel Bagein, Jean Quilbeuf and Louis-Marie Traonouez for the help that they provided me and for the work we accomplished together. I also sincerely thank the members of my jury, Axel Legay, Charles Pecheur, Yves Deville, Luis-Fernando Mejia, Renaud De Landtsheer and Peter Van Roy for the time they spent reviewing my work and for giving me valuable feedback about it. Discussions with all of them highly contributed to improve this thesis.

Going to work was everyday a pleasure, and it is also due to the wonderful colleagues I have. I warmly thank my two office mates: Christophe, who greatly helped me since the beginning of this thesis, and Guillaume, who often reminds me that, sometimes, I have to stop working in order to take lunch together. Big thanks to all INGI department, and special thanks to my BeCool fellows, for the interesting discussions and the fun moments we had in the department or, abroad in conference. Special-special thanks to Ratheil, John and Sascha, as you know... No pain no gain !

I would also like to thank all the people who have helped me indirectly. I think especially about my family and my close friends for their encouragements to whom I could work (and then relax) in excellent conditions.

Last, but certainly not least, I heartfelt thank Lydiane for all the support and the trust she gave me. Making important decisions in life is not something easy but, together, we managed to do it.

# Foreword

This research is financed by the Walloon Region as part of the Logistics in Wallonia competitiveness cluster.

Railway operators are faced with competition from road, air and maritime transport. They need to improve in terms of the globalisation of traffic and the interoperability between operations and infrastructure. To deal with this challenge, Walloon Region (Belgium) initiated in April 2014 a project, called Inograms. Its goal is to maintain the competiveness of railway industry in the face of other transportation means. Given the large scope of this project, it is divided into seven work package, each of them being dedicated to a particular aspect of railway transportation.

This technology exploration is realised in the context of rail interoperability and internationalisation. More specifically, this thesis is related to the first work package which aims to propose innovative solutions for easing the future development of new interlocking systems. The objective pursued was the development of a tool which can be used to automatically generate a new class of interlockings. No safety or availability issue and a maximal fluidity were the requirements of the first work package and are the challenges addressed by this thesis.

# Contents

# Chapter 1
# Introduction

*"What is not started today is never finished tomorrow."*

–Johann Wolfgang von Goethe

Railway is one of the first means of mechanised mass transport. Even now, it is still the backbone of transportation in many countries all around the world. At the beginning, railway networks were only composed of a basic infrastructure designed for a few trains only. At this time, the operations could be managed manually by specialised people, the railway operators, having the duty to regulate the traffic safely and efficiently. However, with the unceasing growth of railway equipments, railway operators must face up to new challenges. The complexity of networks and the density of the traffic have reached such a level that operations can not be handled entirely manually without resorting to computer based methods. A direct consequence of this growth is that the manual decisions quickly become hazardous and can have unpredictable effects on the traffic such as delays, financial losses or worse, accidents. In this context, the need of reliable and efficient methods to assist railway operators is crucial.

Responsibilities of railway operators can be divided into three requirements of a decreasing level of priority. On the one hand, they must ensure the safety and the availability of the traffic and on the other hand, they have to maximise its fluidity. As for any transportation means, *safety* is the most important requirement to consider. It ensures that no accident and no event that could damage people or the infrastructure can occur. Once the traffic is safe, it must also be available in every situation. In other words, trains cannot be stopped in the station without possibility to move. It is related to the *availability* of the traffic. Finally, once the safety and the availability are ensured, the traffic has to be as fluid as possible in order to minimise the total travel time of each train. This aspect is called *fluidity*. Unlike

the safety and the availability that fall under the field of verification, fluidity is related to optimisation where an objective function has to be maximised.

Such responsibilities are in the scope of two railway subsystems: the interlocking and the traffic management system. An interlocking is the subsystem that is responsible for ensuring a safe and available train traffic by controlling active track components of a station. Among these components, there are the signals, defining when trains can move, and the points (also called switches), that guide trains from track to track. Modern interlockings are computerised systems composed of a generic software taking as input data, called application data, describing the actions that the interlocking must take for each situation that can occur in a particular station [TAV09]. The main requirement to consider when designing an interlocking is the safety. A correct interlocking must never allow critical situations such as derailments or collisions. To this purpose, an interlocking must satisfy the highest safety integrity level as stated by Standards EN 50128 [CEN01] and EN 50129 [CEN03] of CENELEC. Beyond the safety, an interlocking must also ensure that no train will be stopped too long in the station in order to maintain the availability of the network.

Operating at a higher layer, the traffic management system controls several interlockings. According to an established timetable, it regulates the traffic in order to ensure the planned traffic. The correctness and the efficiency of both subsystems are then a critical concern for the safety, the availability and the fluidity of the railway traffic.

Analysing and improving such aspects of the railway transportation is the goal pursued by the first work package of Inograms Project, initiated in April 2014 by the Walloon Region (Belgium). It aims to study new technologies to increase the competitiveness of rail operators in the face of transportation means such as aircraft and road transport. More information about Inograms is proposed in Appendix A.

## 1.1 Research Goals

This thesis is carried out within the aforementioned context. Current and state of the art technologies used to design and operate such systems have

some shortcomings and may not satisfy the three requirements in some situations. The bottleneck of an interlocking system is the application data. Although the generic software is developed in accordance with the safety and availability requirements, the reliability of an interlocking is also dependent of the correctness of its application data which are particular to each station. However, preparation of application data is still nowadays done by tools that do not guarantee the required level of safety. Furthermore, the verification of their correctness, as well as their validation, is mainly done manually through a physical simulator that reproduces the behaviour of the interlocking on real infrastructures. In addition to the high cost of this process, it is also error prone because there is no guarantee that all the situations that could end-up in a safety issue have been tested by the simulator.

To overcome this lack, research has been carried out in order to improve this verification process [VHP14, Win02, Eis99, HK02]. Most of it is based on model checking [CKNZ12]. The goal is to perform an exhaustive verification of the system. It is done in three steps. First, the application data and the station layout are translated into a model reflecting the interlocking behaviour. Secondly, the requirements that the interlocking must ensure in order to prevent any safety issue are formalised. Finally, the model checker verifies that no reachable state of the model violates the safety requirements. The main advantage of this method is its completeness: if a requirement is not satisfied, the model checker will always detect it. However, this method suffers from the state space explosion problem. The number of reachable states grows exponentially with the size of the model and the model checker might not return a result within a reasonable time in practice.

Concerning the traffic management system, it has the duty to regulate the traffic in order to maximise its fluidity. However, in case of real time perturbations, this task is handled manually and the actions to perform are decided by human operators. When the traffic management system has predicted a future conflict, railway operators analyse the situation and evaluate the possible actions to do in order to minimise the consequences of the perturbation. Depending on the situation, they can perform some actions (stopping a train, changing its route, etc.) or do nothing. Under stress and given the complexity of the situation, the operators may take suboptimal decisions. Several works already tackle this problem. A recent survey (2014) initiated by Cacchiani et al. [CHK$^+$14] recaps the different trends on models and algo-

rithms for the management of real-time railway disturbances. Most of them are based on Mixed Integer Programming [CGD09, FLM+14, S. 83, LM15]. However, the performance of Mixed Integer Programming models for solving scheduling problems is known to be highly dependant of the granularity of time chosen and is not as flexible for the modelling than other approaches such as Constraint Programming.

Both interlockings and traffic management systems have then shortcomings and can be improved, even the most recent ones. Such improvements are precisely the topic of this thesis. As previously shown, the satisfaction of each requirement is sometimes compromised or is often obtained through a long and tough process. The ambition of our work is twofold: improving the state of the art solutions and providing automatic methods that can be used in practice independently of the size and the complexity of the railway network considered.

## 1.2 Overview of the Contributions

The contributions of this thesis are multiple and can be classified into four categories.

**Contributions Related to Safety** The first contribution proposes to use our knowledge of the railway field in order to accelerate the verification of safety. Concretely, a polynomial algorithm verifying that an interlocking will never cause accidents, given a constraint of monotonicity, is introduced. From another point of view, the second contribution is a proof which can be used in order to prune the search space of a verification based on model checking. A search space growing up polynomially with the size and the complexity of the station is proposed.

**Contributions Related to Availability** This category encompasses the contributions aiming to verify the availability of the traffic against misconfigured interlocking. Several contributions are introduced:

- A model instantiating the behaviour of the application data and the topology of a station. The model is designed in order to be simulated in a discrete event fashion.

- A discrete event simulator which can be run on top of this model.
- Two availability properties that an interlocking must face in order to ensure that no train would be stuck in the network without possibility to move.
- The instantiation of the availability requirements in the aforementioned model and their verification using Statistical Model Checking.

**Contributions Related to Fluidity** Unlike the previous contributions which are oriented to verification, these ones are related to optimisation. The goal pursued is to provide a decision support tool in order to assist railway operators in their decisions. The contributions are as follows:

- A Constraint Programming model for rescheduling the traffic through real time disturbances on the railway network.
- The application of state of the art scheduling algorithms and relevant global constraints in order to achieve a better propagation and a faster search. Concretely, the conditional time-intervals introduced by Laborie et al. [LR08, LRSV09] as well as their dedicated global constraints have been used.
- The formulation of an objective function aiming at the same time to minimise the total delay and to maximise the overall passenger satisfaction. Furthermore, the objective function also considers the heterogeneity of the traffic and different levels of priority between trains through a defined lexicographical order. For instance, a freight train has a lower priority than a passenger train. To the best of our knowledge, there is no work on real time train management dealing with such an objective function.

**Implementation and Technical Contributions** The aforementioned contributions could only be done thanks to the utility tools developed in order to process input data and to analyse them. Concretely two input data are required: the application data and the topology of the network. However, both data have a format not directly exploitable. The first tasks were then to extract and to convert them into an adapted format.

The first technical contribution is a translator designed to automatically parse application data based on SSI [Cri87]. It is a kind of interlocking that

is used by the Belgian railways since 1992. Detailed explanations about this format is provided thereafter. Following the same idea, the second contribution is a parser tool extracting the topology of a station from a data source based on an extension of railML [NHSK04]. Finally, the global technical contribution is a software that has two purposes: verifying automatically the correctness of an interlocking system and optimising the decisions of railway operators in face of real time perturbations in the network.

All the contributions have been tested on three stations of the Belgian railway network: Namêche, Braine l'Alleud and Courtrai. They are presented in Appendix B.

## 1.3 Publications

Aforementioned contributions lead to four main publications:

1. A first version of the model and the discrete event simulator for a global verification of interlocking systems have been published and presented in October 2015 at the 29th European Simulation and Modelling Conference, that was located in Leicester, United Kingdoms [CLSL15].
2. The dedicated algorithm and the proof of monotonicity have been published and presented in September 2016 at the 35th International Conference on Computer Safety, Reliability and Security, that was located in Trondheim, Norway [CS16].
3. An extended version of the previous model and its verification with Statistical Model Checking have been published and presented in January 2017 at the 18th IEEE International Symposium on High Assurance Systems Engineering, that was located in Singapore [CLS+17].
4. The Constraint Programming model aiming to maximise the fluidity of traffic has been published and presented in June 2017 at the 14th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming, that was located in Padova, Italy [CS17]. It also has been presented in June 2017 at the *Treizièmes journées Francophones de Programmation par Contraintes*, that was located in Montreuil-sur-Mer, France.

Let us also mention that this thesis is not a simple aggregation of these publications. It takes rather the most meaningful results in order to provide

a complete solution for the verification and the optimisation of railway interlocking systems. Furthermore, the implementation done has been used for two other publications:

1. Busard et al. have used the translator in order to build a complete framework for interlocking verification with model checking methods. This work has been published and presented in June 2015 at the 4th International Workshop on Engineering Safety and Security Systems, that was located in Oslo, Norway [BCL+15].
2. Following the same philosophy, Limbrée et al. also have used the translator for generating a model which can be verified using a compositional approach based on model checking. This work has been published and presented in June 2016 at the 1st International Conference on Reliability, Safety and Security of Railway Systems, that was located in Paris, France [LCPT16].

## 1.4 Outline

The thesis is divided into four main chapters. With the exception of Chapter 2, each of them propose an innovative solution to the challenges related to safety, availability and fluidity:

- Chapter 2 establishes the background related to the railway traffic management. It describes the structure of an interlocking, its behaviour, the input data considered and how the operations are managed.
- Chapter 3 focuses on safety verification. Most of the research dealing with this challenge is limited to small or medium railway networks because of the state space explosion. This chapter addresses this issue by proposing a polynomial dedicated algorithm for an exhaustive verification of safety. It describes the algorithm, as well as its underlying concepts.
- Chapter 4 deals with the availability requirements of an interlocking. The four main steps of the verification are depicted:
    1. The construction of the model from the application data and the railway topology.
    2. Its execution through a discrete event simulator.

3. The formalisation of the availability requirements that an interlocking must satisfy.
4. The verification of these requirements using Statistical Model Checking.

- Chapter 5 investigates how the fluidity of the traffic can be optimised. It presents the Constraint Programming model and compares its performance with the greedy strategies used by railway operators.

A review of the literature is established for the different aspects. Furthermore, for each solution proposed, a comparison with current and state of the art methods, as well as experimental results based on realistic instances, are also provided. Finally, the last chapter summarises the main conclusions and results obtained and investigates some directions for future works.

# Chapter 2
# Interlocking Principles

*"The introduction of so powerful an agent as steam to a carriage on wheels will make a great change in the situation of man."*

–Thomas Jefferson

## 2.1 Context

In the railway domain, an *interlocking* is an arrangement of systems that prevents conflicting train movements in a station by controlling its active track components. It is more specially a signalling subsystem that acts as an interface between the traffic and the components. It is done in two steps. First, the interlocking collects information about the occupation of the track layout and about its movable components. When achieved, it then evaluates the information and can permit or refuse train movements by setting signals on a proceed state or letting it by default on a stop state. Interlocking functions can be split into three hierarchical levels [TAV09]:

1. The *operational level* acts as the interface between the signalman performing the request and the machine.
2. The *interlocking level* includes the functions required to decide if the request performed by the signalman can be accepted, and to react consequently.
3. The *element control level* mainly operates on the hardware by defining the functions required to transmit information between the components.

Historically, each function was performed by a human. However, with the unceasing growth of railway networks, this solution quickly became ineffective. Over the years, the progression of interlocking technology kept increasing. This evolution is depicted in Figure 2.1.

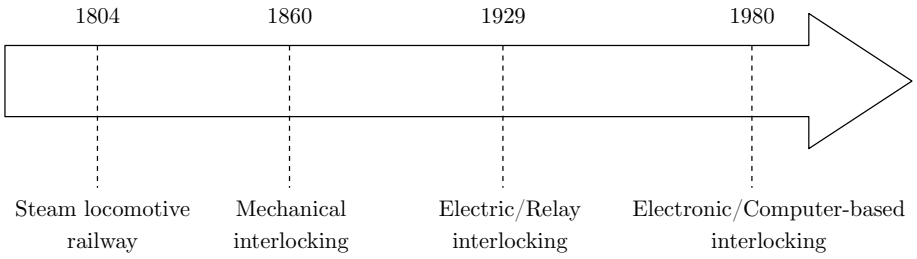



**Figure 2.1.** Evolution of the railway interlocking technology [McN02, TAV09].

In a *mechanical interlocking*, the railway operator controls the system through mechanical levers which are interconnected by wires. Contrariwise, *electric interlockings* are manipulated through electric buttons. The physical components are also controlled electrically using a relay technology. *Electronic interlocking*, also called computer-based interlocking, is the most recent technology and is currently the most used in the world. All the functions are performed electronically with a computer hardware and software. The software is generally composed of a generic and a specific module. A large range of electronic interlockings exist: SSI [Cri87], SMILE [AWNO85], ACC [AIM+96], Ebilock [TAV09], etc. Each of them is dedicated to a specific case which usually depends on national requirements, each country having its own procedure and rules. However, SSI and SMILE are generally considered as the two standardised forms of electronic interlockings.

This thesis deals with the specific module of computer based interlockings. Whereas the generic module has to be developed only once, the specific module depends on the railway network considered and must then be designed for each interlocking. The safety, implying that no accident will occur, is the most important aspect to consider. European Railway Agency has edited strict safety norms in an effort to harmonise the signalling principles and rules at the European level [GR14, CEN01, CEN03]. Although the generic software is developed in accordance with these norms, the correctness of an interlocking also relies on the correctness of the specific module. Furthermore the availability aspect must also be considered.

Currently, the specific module is prepared manually and is thus subject to human errors. For example, some prerequisites to the clearance of the

signal can be missing. This kind of error can easily be discovered by a code review or a static analysis [Liv06]. However, errors caused by concurrent actions, like route commands, are much harder to spot. In this case, the number of possible concurrent actions explodes quickly and testing manually all the combinations is impracticable. As an exhaustive testing is impossible, the manual validation of the application data relies on a relaxed verification process:

1. Firstly, the *functional tests* ensure that the system responds properly to the commands issued by the controller. Those tests are performed by the expert who wrote the specific module.
2. Secondly, the *safety tests* check that each command is tested and that all the conditions that are supposed to impact the commands are evaluated in all their possible values. Those tests are prepared and carried out by an independent checker.
3. Finally, the specific software is approved by the engineer in charge of the project through a *reviewing* step.

During this process, all the anomalies are traced in a bug management tool and must be fixed before the interlocking is commissioned. This validation is mainly done manually through a physical simulator that reproduces the behavior of a real environment and applies it on interlocking machines. In addition to the high cost of this process, it is also error prone because there is no guarantee that all the situations that could end up in a safety or an availability issue have been tested by the simulator.

The specific module defines the actions that can be done by an interlocking system and under which conditions. Such information can be defined in different ways according to the type of interlocking considered. Most interlockings are *route based*. Informally, a route is the path that a train is supposed to follow inside a station. In general situations, when a train is entering a station, a railway operator assigns a route to the train by performing a request. The interlocking then processes the request, decides if it can be accepted and finally performs the required actions in order to fulfil the request. Otherwise, the request is aborted.

## 2.2 Railway Components

An interlocking system acts as an interface between the traffic controller and the components of the railway network. The main components are represented in Figure 2.2 which illustrates the track layout of Braine l'Alleud, a medium-sized station of the Belgian network (see Appendix B.2 for more details). Components can be either physical or logical.

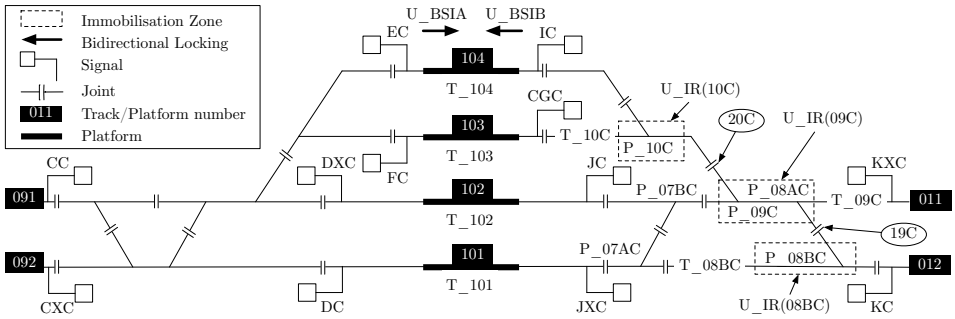



**Figure 2.2.** Layout of Braine l'Alleud Station.

*Physical components* have a real representation in the layout. They are as follows:

- The *tracks* (e.g. `091`) are the railway structure where trains can move. A track can be a *platform* (e.g. `104`) if trains can stop on it in order to pick up passengers.
- The *track segments* (e.g. `T_08BC`) are the portions of tracks where trains can be detected. They are delimited each other by the *joints* (e.g. `19C`).
- The *points* (e.g. `P_07BC`), also called railway switches, are the movable devices that allow trains to move from one track to another. According to Belgian convention, they can be in a normal position (left) or in a reverse position (right).
- The *signals* (e.g. `CXC`) are the devices used to control the train traffic. They are set on a proceed aspect (e.g. green) if trains can safely move into the station or in a stop aspect (red) otherwise. They are also used

for displaying other information to the train driver like the authorisation to proceed but with a limited speed.

These components are controlled and monitored by a single interlocking. For instance, it can detect that a train is waiting on Track `011` in front of Signal `KXC` and then puts it on a proceed aspect if this action is safe.

On the other hand, a route based interlocking also uses *logical components*. Unlike physical components, they are intrinsic variables of the software and do not have a real representation in the layout. The main component is the *route*. It is the path that a train is supposed to follow inside a station. It is generally named according to their origin and their destination. Signals are often used as references for the origins whereas tracks or platforms are commonly used for destinations. For instance, Route `R_CXC_101` starts from Signal `CXC` and ends on Platform `101`. Besides, other logical components are used:

- The *subroutes* are the contiguous segments that the trains must follow inside a route. For instance, `R_KC_102` is split into subroutes from `KC` to `19C`, from `19C` to `20C` and from `20C` to `CGC`.
- The *immobilisation zones* (e.g. `U_IR(09C)`) are the variables materialising the immobilisation of a set of components, typically points. When they are locked, the attached point cannot be commanded. For instance, `U_IR(09C)` prevents Points `P_08AC` and `P_09C` to be commanded.
- The *bidirectional locking* is the mechanism used to prevent head to head collisions on platforms. Each bidirectional locking consists of two variables: `U_BSIA` and `U_BSIB`.
- The *fictive signals* (none in Braine l'Alleud but `S_f_5D_B` in Courtrai, Appendix B.3) are particular signals that have no physical structure. They are used by the interlocking in order to split a full path in a sequence of routes. For instance, a possible full path from Signal `MD` to Track `205` in Courtrai is the sequence of Routes `R_MD_5D` and `R_5D_205` which are split by Fictive Signal `S_f_5D_B`. Concept of full path is described in more detail in Chapter 3.

Both physical and logical components are then used by the interlocking in order to control the traffic. More details about how it is used in practice is provided thereafter.

## 2.3 Solid State Interlocking

The work presented in this thesis is based on Solid State Interlocking (SSI) technology. SSI refers together to the first computer-based interlocking systems, to the software used to develop these systems and to their programming language. Its development started in the 1970s by British railways in collaboration with GEC (now Alstom) and Westinghouse. It quickly became one of the most widely used interlocking systems, especially in Western Europe and countries of the British Commonwealth [TAV09]. It is used in Belgium since 1992. The software is structured into two modules:

1. The *generic software* which serves as an interpreter for the application data and carries out system functions like communication. It does not change from one station to another and can then be verified only once. Its development and validation follow the highest safety rules applicable to the domain (Standards EN 50128 [CEN01] and EN 50129 [CEN03] of CENELEC).
2. The specific *application data* which express all interlocking functions. They are written in a data language designed to be used by signalling engineers and operators. Unlike the generic software, application data are specific to a station and must then be verified individually for each station considered.

The ability of an SSI-based interlocking to avoid critical situations, like train collisions, relies on the safety level achieved by the combination of both modules. To do so, application data use several variables, instantiating both physical and logical components. They are depicted in Table 2.1.

Each component has then a unique identifier and one value defining the status of the component. For instance, `P_01BC cdn` indicates that Point `P_01BC` is directed to the normal position. Furthermore, the whole set of application data is divided into several configuration files. They are as follows:

- The *panel route request* file (PRR) defines the conditions that must be satisfied for a route request acceptance. The actions that the interlocking must perform when the request is accepted are also defined inside.

| Component | Value | Meaning |
|---|---|---|
| Immobilisation zone<br>Subroute<br>Bidirectional locking | `f`<br>`l` | Free<br>Locked |
| Route | `s`<br>`xs` | Set<br>Unset |
| Track segment | `c`<br>`o` | Clear<br>Occupied |
| Point (condition for moving) | `cfn`<br>`cfr` | Condition for normal position<br>Condition for reverse position |
| Point (position) | `cdn`<br>`cdr` | Directed to normal position<br>Directed to reverse position |
| Point (action) | `cn`<br>`cr` | Command to normal position<br>Command to reverse position |
| Signal | `stop`<br>`proceed` | Red aspect<br>E.g. green aspect |

**Table 2.1.** Application data components in SSI format.

- The *points free to move* file (PFM) includes all the conditions that must be ensured before moving a point to its normal and its reverse position.
- The *flag operation* file (FOP) defines the necessary conditions for freeing a component.
- The *output file* (OPT) contains the outputs of the interlocking, as the description of the life cycle of a route from their command to their releasing. It is used to know when the start signal of a route can be set at a proceed state.

The contents of such files, as well as the definition of variables, are formalised by grammars, which are available in Appendix C. Examples are provided in the next section. The grammars are based on ENBF Standard [Sco98] and are context-free [Gin66] according to Chomsky hierarchy [Cho56]. They can be efficiency parsed with different algorithms such as Early [Ear70], CYK [LL09], Packrat [For02] or recursive descent [Rei]. The choice of the algorithm is dependent of specific properties that the grammar must satisfy. Let us finally mention that the grammars proposed cover only a subset of the

application data, which is relevant for the interlocking correctness. The other information serve different purposes such as the interlocking communication and are not related to safety or availability.

## 2.4 Interlocking Behaviour

As previously said, a route based interlocking controls active components of the station in order to ensure a safe and available traffic. For SSI based interlocking, all the possible actions and their underlying conditions are described in the application data. To illustrate this behaviour, let us consider a scenario where a train is coming from Track `012` and has to go to Platform `103` in Braine l'Alleud (Figure 2.2). The process follows different steps.

Firstly, when the train is waiting at Signal `S_KC`, the interlocking verifies whether the request for Route `R_KC_103` can be granted. Listing 2.1 presents the application data related to this request.

```
1  *Q_R(KC_103)
2     if     R_KC_103 xs,
3            P_08BC cfr, P_08AC cfr, P_09C cfr, P_10C cfn,
4            U_IR(08BC) f, U_IR(09C) f, U_IR(10C) f
5     then   R_KC_103 s,
6            P_08BC cr, P_08AC cr, P_09C cr, P_10C cn,
7            U_IR(08BC) l, U_IR(09C) l, U_IR(10C) l,
8            U_KC_19C l, U_19C_20C l, U_20C_CGC l
```

**Listing 2.1.** Request for commanding Route `R_KC_103`.

The request is accepted only if Route `R_KC_103` is not already set (line 2 - each route request has a similar condition), if relevant points are free to be commanded to the reverse (`cfr`) or normal (`cfn`) position (line 3) and if some immobilisation zones are not locked (line 4). If all the conditions are satisfied, `R_KC_103` is set (line 5), the points are controlled to the reverse (`cr`) or normal (`cn`) position (line 6) and some components as the immobilisation zones (line 7) and subroutes (line 8) are locked. At this step, Route `R_KC_103` is set, or *commanded*, but not yet *proved*. A route is proved only when its start signal turns on a proceed aspect. However, the start signal is still on a stop aspect and the train can thereby not enter in the station yet. Before moving a point, the interlocking must verify that this action can safely be

executed. Listing 2.2 illustrates such conditions for Point P_08AC.

```
1  *P_08ACN U_IR(09C) f
2  *P_08ACR U_IR(09C) f
```

**Listing 2.2.** Conditions allowing Point P_08AC to move.

Directly after the acceptance of the request in Listing 2.1, the interlocking tries to lock a bidirectional locking in order to prevent routes going to Platform 103 from the left to be proved. It is shown in Listing 2.3.

```
1  if U_BSIA(103) f then U_BSIB(103) l
```

**Listing 2.3.** Request for setting the bidirectional locking of Platform 103.

Once R_KC_103 has been commanded, the interlocking will check if it can safely prove the route and so gives the train an authority to move.

```
1  *R_KC_103
2     if     P_08BC cdr, P_08AC cdr, P_09C cdr, P_10C cdn,
3            U_IR(08BC) l, U_IR(09C) l, U_IR(10C) l,
4            T_08BC c, T_09C c, T_10C c, T_103 c,
5            U_BSIA(103) f
6     then   U_BSIB(103) l, S_KC proceed
```

**Listing 2.4.** Request for proving Route R_KC_103.

Listing 2.4 states that R_KC_103 can be proved only if the points are commanded and detected in the requested position (cdn and cdr on line 2), if the immobilisation zones are locked (line 3), if there is no train on relevant track segments (line 4) and if the bidirectional locking for trains coming from right to Platform 103 is free (line 5). The route proving results on locking the paired bidirectional locking and on setting Signal S_KC on a proceed state (line 6). At this step, the train can finally move into the station.

When not used anymore, locked components can be released. It is done according to the progress of the train on its route. After each train movement, the interlocking checks if a releasing event can be triggered. Listing 2.5 states the conditions for releasing Subroute U_20C_CGC. If all the conditions are fulfilled, the requested components are released.

```
1 U_20C_CGC f if U_KXC_20C f, U_19C_20C f, T_10C c
```

**Listing 2.5.** Conditions for releasing Subroute `U_20C_CGC`.

This process briefly describes the life cycle of a route and how it is managed by the interlocking. Errors in application data can lead to disastrous situations. For instance, if the bidirectional locking is not properly checked before proving Route `R_KC_103` (line 5 missing from Listing 2.4), two routes going to the same platform from a different side can be proved together which will potentially cause a head-on collision. It is why the application data correctness is a critical concern.

## 2.5 Station Topology

The previous sections introduced the application data, which define the behaviour of an interlocking for a specific station, but which contain no information related to the track layout of the station. However, the correctness of an interlocking is also dependent on its consistency with the track layout. For this reason, a reliable data source describing the track layout is required. It can be encoded in different ways. A common approach is to represent the topology using a computer-aided design such as AutoCad [EO+98] and to encode different information such as the track lengths or the geographic position of the components. It is the method currently used in Belgium, but it has some shortcomings. Given that the information is presented graphically as a drawing, the processing of the schema and its integration into a verification model require manual work and can then hardly be automated.

A second method is to use a structured language, which eases an automatic processing. For instance, *railML* [NHSK04] is a markup language specialised for the railway domain. It was conceived to give a universal support for information which can be used for any application related to the railway field. It is used by several companies such as Alstom, Siemens, Bombardier, Thales or Toshiba. It is structured with four main schemas that are as follows:

1. The *Infrastructure*, which contains information about the railway network topology and the physical components.

2. The *Timetable and Rostering*, which contains information related to the timetables and the schedules.
3. The *Rolling Stock*, which takes over all the information about the vehicles used.
4. A last schema which encompasses all the remaining information not included in the previous schemas.

Only the infrastructure schema is relevant for our purpose. However, the current version of railML[1] does not include yet enough information in order to fully define the topology of a network. For this reason, an extension of railML, called *railML+*, is proposed. Two pieces of information are added: the position of the joints, which separates the track segments and the definition of the routes that can be set inside the topology. Thanks to its completeness and its extensibility, railML+ can specify the track layout of a station with more flexibility than the graphical approaches. Furthermore, such a format can be easily parsed using different algorithms [LDL+08]. For these reasons, it is the format that we used.

## 2.6 Operational Decisions

Beyond the interlocking operations, the railway traffic must also be regulated according to an established timetable. It is under the responsibility of the *traffic management system*. The first step is to define the schedule. While earliest schedules could be built manually without using computer based methods, it is not possible anymore. Plenty of works deal with this problematic of building the most appropriate schedule for general [HKF96, GSA04, ZZ05], or specific purposes [HS16, SWDK15].

However, practical schedules must also deal with real time perturbations. Disturbances, technical failures or simply consequences of a too optimistic theoretical schedule can cause delays which can be propagated on the traffic. The initial schedule may then become infeasible. Real-time modifications of the initial schedule therefore may be required. In this case, traffic management is often handled manually and the actions to perform are decided by human operators. We refer it as the *traffic rescheduling problem*. The procedure follows this scenario:

[1] Version 2.3, released on March 9th, 2016.

1. The traffic management system has predicted a future conflict caused by perturbations on the traffic.
2. The operators controlling the traffic analyse the situation and evaluate the possible actions to do in order to minimise the consequences of the perturbations. Besides the safety and the availability, the criterion considered for the decision is the fluidity, which has to be maximised. It often consists in minimising the sum of delays of trains pondered by their number of passenger, taking into account that some trains can have a higher priority than others.
3. According to the situation, they perform some actions (stopping a train, changing its route, etc.) or do nothing.

Railway operators must deal with several difficulties in order to tackle the traffic rescheduling problem. Firstly, the available time for analysing the situation and taking a decision can be very short according to the criticality of the situation. Time to take a decision can be for instance less than one minute. Furthermore, for large stations with a dense traffic, particularly during the peak hours, the effects of an action are often difficult to predict. Finally, the number of parameters that must be considered (type of trains, number of passengers, etc.) complicates the decision.

Such reasons can lead railway operators to take decisions that will not improve the situation, or worse, will degrade it. Most of the time, the decision taken is to give the priority either to the first train arriving at a signal, or to the first one that must leave the station [DPP07]. However, it might not always be the best decision. It is why the use of a decision support tool based on optimisation in order to assist operators in their decisions is advocated.

# Chapter 3
# Safety Verification

*"Safety isn't expensive, it's priceless."*

–Anonymous quote

## 3.1 Motivation

The main requirement to consider when designing an interlocking is *safety*. A correct interlocking must never allow critical situations such as derailments or collisions. To this purpose, an interlocking must satisfy the highest safety integrity level as stated by Standards EN 50128 [CEN01] and EN 50129 [CEN03] of CENELEC. Although the generic software is developed in accordance with these requirements, the safety of an interlocking is also dependent of the correctness of its application data which are specific for each station. As previously indicated, the preparation of the application data is still nowadays done by a process that does not guarantee the required level of safety. Most of them are prepared manually and are thus subject to human errors.

To tackle this problem, research has been carried out in order to improve the current verification process. A plethora of methods are proposed in the literature. For instance, Hartonas-Garmhausen et al. [HGCC+98] propose a verification based on real time constraints. They use Verus [CC97] for the modeling and express the properties as invariants in a computational tree logic. Moler et al. [MNR+12c, MNR+12a, MNR+12b] propose to use CSP‖B [BL05] for the verification. The overall specification combines two communicating models: a description of CSP processes and a collection of B machines. B Language [Lan12] is also used by Abo et al. [AV13] which show how OVADO tool can be used for the verification. Besides, some authors consider

Petri nets technology for reasoning about the problem [VA10, AA08, SCdB15].

Most of the methods are based on model checking. It is corroborated by the recent surveys of Fantechi et al. [FFM13] and Haxthausen et al. [HNR16] which recap some trends for the verification of interlocking systems. However, model checking suffers from the state space explosion problem [CKNZ12]. The number of reachable states exponentially grows as the size of the model grows and the model checker might not return a result within a reasonable time in practice. Although it is possible to verify interlockings on small or medium areas, this approach has some difficulties to scale for larger stations such as Courtrai (Appendix B.3). The main challenge for model checking is then to deal with the state space explosion problem.

To do so, Winter et al. [WJR+06] suggest to keep the model as simple as possible by abstracting some parameters, such as the speed of trains or their length. Besides, improvements can also be done intrinsically on the model checking algorithms. Different studies propose to use symbolic model checking instead of explicit approaches [Eis99, HK02]. Smart variable ordering can also be considered in order to speed up the verification [Win12]. In [VHP14], Haxthausen et al. detail how an ETCS level 2 compatible Danish interlocking can be modelled. The state space, the transition relations and the safety properties are efficiently evaluated by solvers based on SMT [BSST09] that support bit vector and integer arithmetic. Bounded model checking [HPP13] or static checking [HØ16] are also considered. Finally, state space reduction can be carried out using partial order reduction techniques [ABH+97, KLM+98, Pel98, FG05, KWG09], as done by Cimatti et al. [CGM+98a, CGM+98b] who use SPIN model checker [Hol97].

For the specific verification of application data expressed in SSI, Huber and King [HK02] have proposed an integrated model checker. They demonstrate how safety properties can be verified automatically. Their tool, gdlSMV, directly reads geographic data and builds a corresponding representation of the data. Finally, the verification is performed using symbolic model checking algorithms with NuSMV [CCGR00]. Compared with this work, Busard et al. [BCL+15] propose a unified and fully automated approach aiming to verify completely the safety of an interlocking system. They developed an automatic generator of the model from the application data as well as custom model checking algorithms with PyNuSMV, a Python library

based on NuSMV [BP13]. Finally, Limbrée et al. [LCPT16] have improved this previous work with a modelling based on a compositional approach and use modern model checking algorithms, such as IC3 or k-liveness for the verification. The idea of a compositional approach is also considered in other works [MFH16, XTGC09].

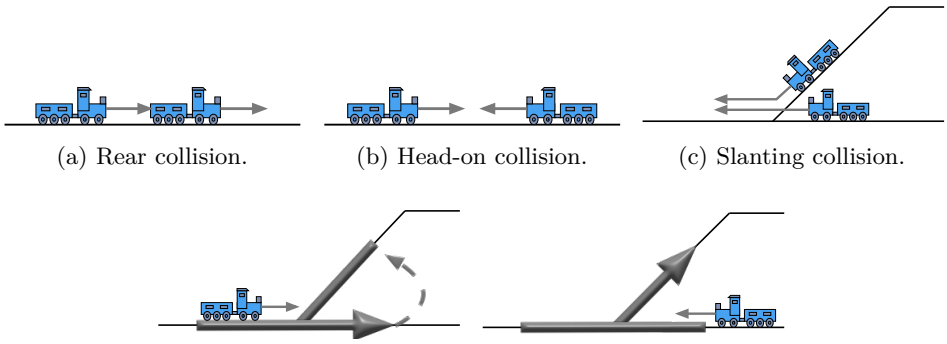
This related work constitutes a large part of the state of the art methods for the automatic verification of railway interlocking systems. However, despite the great progresses achieved, automatically verifying large stations is still a challenge. It is within this context that the first part of our research is based. The objective pursued is to design a scalable method for verifying the correctness of the application data in terms of safety. This chapter is structured into four parts:

1. The requirements that any interlocking system must satisfy to ensure the safety are formulated.
2. The main contribution for the safety verification is presented. Concretely, we propose a state space which grows polynomially in function of the problem size and which relies on a proof of the application data monotonicity.
3. The application of this proof is then used for the specific case of the application data expressed in SSI. It gave birth to a polynomial algorithm for the safety verification.
4. Experimental results showing the adequacy and the scalability of the approach are presented.

## 3.2 Safety Requirements

The main goal of an interlocking is to ensure the safety of the train traffic. To do so, it must prevent the traffic of any accident that can occur in the controlled area. The first step is then to define what are the possible accidents and how they can be detected. Generally speaking, an accident or a *safety issue*, corresponds to any situation violating the safety and that leads to human or material damages. Different authors [Win02, Anu09, TRN$^+$02] identified two types of safety issues: collisions and derailments. There exist three kinds of collisions (rear, head-on and slanting collision), but all of them are characterised by the same effect: two trains are in the same place at the

same time. Furthermore, derailments can occur in two situations. On the one hand, it occurs when a point is not set on a position allowing trains to stay on the railway structure. On the other hand, a point moving when a train is on it also causes a derailment. Figure 3.1 illustrates the five situations leading to an accident.



(a) Rear collision. (b) Head-on collision. (c) Slanting collision.

(d) Derailment (case 1). (e) Derailment (case 2).

**Figure 3.1.** Situations leading to an accident due to an interlocking malfunction.

According to Busard et al. [BCL$^+$15], there are three requirements that must hold in order to avoid safety issues.

**Requirement 1** (No collision)**.** *A same track segment cannot have two trains or more on it at the same time. Otherwise, a collision will occur.*

**Requirement 2** (Consistent locking)**.** *A point cannot move if there is a train on it. Otherwise, the train will derail.*

**Requirement 3** (Continuous path)**.** *A point must always be set on a position allowing trains to continue their path in order to avoid derailments. For instance, Point* `P_07BC` *in Figure 2.2 must be set to the reverse position (right) for trains coming from Signal* `JC`*. If such a property is not satisfied, the train will derail.*

Such requirements can be expressed by means of *invariants*. They have to be satisfied in all the reachable states of the system. Beyond the safety, an implicit requirement is that a train must always reach its planned destination.

**Requirement 4** (Consistent path)**.** *A train must always reach its planned destination.*

## 3.3 Search Space Pruning

An important part of the related work does not take advantage of the intrinsic specificities of the considered system for its verification. Indeed, the improvements they propose remain generic and although they can be applied for any model checking application, they do not use knowledge of the railway field in order to ease the verification. Our approach tackles the problem with a different perspective. Instead of limiting our knowledge of the system only for its modelling, we use it for the verification itself.

Specificities of the system can be used to identify what are the scenarios that can lead to safety issues and to distinguish them from others that are either redundant or that never happen in practice. The state space covering all the possible situations is then pruned and the verification is faster. An automatic and exhaustive verification is still performed, but compared to previous methods based on model checking, it is restricted to a limited state space that increases polynomially in function of the number of routes and track segments.

Classical model checking approaches, such as [BCL$^+$15, LCPT16, HK02], often consider a state space defined in terms of routes. All the combinations of routes with their own whole life cycle are tested which leads to an exponential growth of the state space in function of the number of routes. The idea behind our method is to verify that no safety issue occurs in any situation by considering only pairs of routes. The correctness of this idea is then based on the assumption that testing only pairs of routes is sufficient for detecting all the safety issues. We argue that this statement is correct provided that the application data are *route-monotonic*. Let us first define the notion of *constrained components set* of a station state.

**Definition 3.1** (Constrained components set)**.** *The constrained components set of a station state ($S_1$) is the set of components ($C_1$) that are currently locked in $S_1$. A station state $S_2$ is more constrained than $S_1$ if and only if we have $C_1 \subseteq C_2$.*

In other words, all the components that are locked in $S_1$ are, at least, also locked in $S_2$. For instance, let us consider the states in Braine l'Alleud (2.2) before ($S_1$) and after ($S_2$) that the command request depicted in Listing 2.1 has been successfully performed. The request triggers the locking of some components (`U_IR(08BC)`, `U_IR(09C)`, `U_IR(10C)`, `U_KC_19C`, `U_19C_20C` and

`U_20C_CGC`) and no releasing. Then, $S_2$ is more constrained than $S_1$. We can now state the concept of route-monotonicity.

**Definition 3.2** (Route-monotonicity)**.** *Application data are route-monotonic provided that if a route cannot be commanded given a particular station state, it will not be able to be commanded for a more constrained station state. The same rule must also apply for releasing instructions.*

In other words, if a route $r_1$ cannot be commanded when a route $r_2$ is commanded, it cannot be commanded if $r_2$ and a third route $r_3$ are commanded together. Application data are not route-monotonic if conditions for route commands (Listing 2.1) require components to be locked instead of being free. It is because the station becomes more constrained each time a component is locked for a route. The same rule also applies for the components releasing such as in Listing 2.5. The property of route-monotonicity can be easily checked through a static analysis of the application data. To do so, one can simply read sequentially the application data and check separately each condition. It is depicted in Algorithm 3.1.

**Algorithm 3.1:** Static verification of route-monotonicity property in application data.

```
1  INPUT: prr, the PRR file of the application data that we want to verify.
2         fop, the FOP file.
3  OUTPUT: True if the application data are route monotonic.
4          False otherwise.
5
   // PRR file analysis (grammar in Appendix C.2)
6  for r ∈ prr.⟨route_request⟩ do
7      for c ∈ r.⟨condition⟩ do
8          if c.⟨value⟩ = 1 then
9              return False
   // FOP file analysis (grammar in Appendix C.4)
10 for r ∈ {fop.⟨subroute_release⟩, fop.⟨UIR_release⟩, fop.⟨UBSI_release⟩} do
11     for c ∈ r.⟨condition⟩ do
12         if c.⟨value⟩ = 1 then
13             return False
14 return True
```

Semantic of $\langle\rangle$-components is described in Appendix C. For all the possible route command requests (line 6), if at least one condition for commanding

a route (line 7) requires component to be locked instead of being free (line 8), then the application data are not route-monotonic. It is also checked for components releasing (lines 10-13). This algorithm has a time complexity linear in function of the size of the application data ($\mathcal{O}(|A|)$ with $|A|$ the size of the application data). The concept of route-monotonicity can then be used to state Property 3.3.

**Property 3.3** (Route-sufficiency)**.** *Considering only pairs of routes is sufficient to verify the safety of an interlocking based on the application data format described previously provided that they are route-monotonic.*

*Proof.* We have to prove that all the requirements can be verified by using at most two routes. An issue (point misdirected - violation of Requirement 3) can occur if the first route is not properly set initially. This case only requires routes taken separately and is then trivially proved. Furthermore, an issue can also occur if the actions related of a second route interact with components already used by the first route. We need to prove that considering two routes is sufficient to detect all possible issues. Let us consider $C$, the set of all the components, either physical or logical, of the station and $C_i \subseteq C$, the set of components used or locked by a route $r_i$. Let us take two arbitrary routes, $r_1$ and $r_2$. There are two possible situations:

- $C_1 \cap C_2 = \emptyset$: the two routes have no component in common and are then completely disjoint. No issue can happen between them.
- $C_1 \cap C_2 \neq \emptyset$: the routes share at least one component. If the interlocking allows both routes to be set together at the same time, an issue can happen.

Other issues (violation of Requirements 1 and 2) can be represented as an intersection between such sets because they are both generated by a same component used by both routes. An intersection is formed by at least two routes. Two routes are then sufficient to detect any issue provided that the command of a third route will not relax $C_1$ or $C_2$ by releasing some components thereafter. According to Definition 3.2, application data must be route-monotonic to avoid that. In this case, testing only the pairs of routes is sufficient to cover all the conflictual scenarios regarding Requirements 1, 2 and 3. □

This kind of assumption is also considered in [MNR+12c] where the verification is limited to two trains. However, in practice, all the application

data are not route-monotonic. It happens when the full path (defined below) of a train is not only determined by a single route but by a sequence of several routes instead. It is for instance the case for Courtrai where a possible path from Signal MD to Platform 205 is determined by Routes R_MD_5D and R_5D_205. The related route command for R_5D_205 is depicted in Listing 3.6.

```
1  *Q_R(5D_205)
2     if     R_5D_205 xs
3            P_15AD cfr, P_15BD cfr, P_16D cfr,
4            P_14BD cfn, P_14AD cfn,
5            U_IR(15BD) f, U_IR(14BD) f,
6            U_5D_A l
7     then   (...)
```

**Listing 3.6.** Request for commanding route R_5D_205 in Courtrai.

In this request, a necessary condition for commanding R_5D_205 is the locking of Subroute U_5D_A (line 6). The purpose of this condition is to allow R_5D_205 to be commanded only if another route, which uses U_5D_A, has been commanded beforehand. This mechanism is currently used when routes do not allow trains to entirely cross the whole station. According to Definition 3.2, Courtrai is not route-monotonic because the command of any route locking U_5D_A is a preliminary condition for commanding R_5D_205. The notion of route-monotonicity has then to be extended. To do so, let us introduce the concept of full path. A *full path* in a station is a sequence of several routes that are commanded successively. It is constituted of a sequence of $n$ routes $[r_1, \ldots, r_n]$ with $n \in \mathbb{N}_{\neq 0}$ having the following property: a route $r_i$ with $i \in [2, n]$ can be commanded only if $r_{1-1}$ is also commanded. Route $r_i$ requires then a more constrained state for its command which is inconsistent with the route-monotonicity property. Let us extend this concept with Definition 3.4.

**Definition 3.4** (monotonicity-complete)**.** *Application data are monotonic-complete provided that if the sequence of routes composing a full path cannot be commanded given a particular station state, it will not be able to be commanded for a more constrained station state. The same rule must also apply for releasing instructions.*

As for the route-monotonicity case, monotonicity-complete property can be checked through a static analysis of the application data (Algorithm 3.2).

**Algorithm 3.2:** Static verification of complete-monotonicity property.

```
1  INPUT: prr, the PRR file of the application data that we want to verify.
2         fop, the FOP file.
3  OUTPUT: True if the application data are monotonic-complete.
4          False otherwise.
5
   // PRR file analysis (grammar in Appendix C.2)
6  for fp ∈ FULL_PATH do
7      lockSet ← ∅ // record the components locked during the command
8      for r ∈ fp do
9          req ← getRelatedRequest(r)
10         for c ∈ req.⟨condition⟩ do
11             if c.⟨value⟩ = 1 and c.⟨variable⟩ not in lockSet then
12                 return False
13         lockSet ← ∅ // refresh the set
14         for a ∈ req.⟨assignment⟩ do
15             if a.⟨value⟩ = 1 then
16                 lockSet ← lockSet ∪ a.⟨variable⟩
   // FOP file analysis (grammar in Appendix C.4)
17 for r ∈ {fop.⟨subroute_release⟩, fop.⟨UIR_release⟩, fop.⟨UBSI_release⟩} do
18     for c ∈ r.⟨condition⟩ do
19         if c.⟨value⟩ = 1 then
20             return False
21 return True
```

Let us define *FULL_PATH* as the set of all possible full paths and *getRelatedRequest(r)* a function returning the request associated to a route $r$. For instance, *getRelatedRequest*(`KC_103`) returns the route request described in Listing 2.1. For each full path (line 6), the related routes are processed sequentially (line 8). The application data are not monotonic-complete if the command of a route requires components to be locked instead of being free, unless if this component has been locked by the command of the previous route composing the full path (line 11). To do so, the components locked during the previous request are recorded in a set (lines 14-16) which is refreshed after each request (line 13). For the components releasing part (lines 17-20), it works as in Algorithm 3.1. This algorithm has a time complexity of $\mathcal{O}(|A| \times r \times fp)$ with $|A|$ the size of the application data, $r$ the maximal number of route composing a full path and $fp$ the number of

full paths. In practice, $r$ is negligible compared to $fp$ (each full path in Courtrai is composed of at most 2 routes). With the same reasoning than for route-sufficiency property, we can infer Property 3.5.

**Property 3.5** (complete-sufficiency)**.** *Considering only pairs of full paths is sufficient to verify the safety of an interlocking based on the application data format described previously provided that they are monotonic-complete.*

*Proof.* We have to prove that all the requirements can be verified by using at most two full paths. The proof is similar than for the route-sufficiency proof. The only difference is the definition of the sets. Instead of considering the set of all the components used by a single route, we consider the set of components of all the routes composing the full path. □

Among the case studies considered, all of them are monotonic-complete but only Namêche and Braine l'Alleud are route-monotonic. So far, we have not found consistent application data where the monotonicity-complete property is not satisfied. We then propose the following conjecture:

**Conjecture 3.6.** *Application data, that are expressed in SSI and that are used in practice, are monotonic-complete.*

The main implication of this conjecture is that the concepts of route-sufficiency and complete-sufficiency can be used in order to prune drastically the search space of a verification based on model checking for any application data expressed in SSI. Finally, a last property of the application data is used (Property 3.7).

**Property 3.7.** *A route cannot be commanded if it is already commanded.*

This property is a consequence that routes are linked to at most one train at a time. In other words, once a route is commanded, it must be completed before the route can be commanded again. It is expressed in the application data as a condition in route commands such as in Listing 2.1 where `R_KC_103` cannot be already set. Such a condition is a preliminary condition for the correctness of the application data and must always hold. As we will see, this property can be used in order to infer a partial order reduction for the verification.

Even if the above concepts and assumptions done for the state space reduction have been defined from application data expressed in SSI, they can be adapted for other route-based interlocking formats.

## 3.4 Verification Algorithm

The aforementioned concepts can be used in order do design a polynomial algorithm for the verification of safety. Basically, this algorithm is related to model checking using an implicit partial order reduction based on our knowledge of the system. The goal is to exploit the equivalence of some states in order to reduce the search space. An automatic and exhaustive verification is still performed, but now the verification is restricted to a limited state space that increases quadratically in function of the number of routes, or full paths, and track segments. Furthermore, only the scenarios that can happen in practice are generated.

Let $S$ be a station. We define $ROUTES$ as the set of all routes, $TRACK\_SEGMENTS$ as the set of all track segments, $POINTS$ as the set of all points and $COMPONENTS$ as the set of all components in $S$. The algorithm returns *True* if $S$ satisfies all the requirements and *False* otherwise. For all routes $r$, we define *r.origin* as the origin of $r$, *r.destination* as its destination, *r.isCommanded* and *r.isProved* as boolean values defining if $r$ is commanded and proved respectively. We also define *t.position* as the current position of a train $t$, *p.state* as the state (normal or reverse) of a point $p$ and *c.isLocked* as a boolean value defining whether a component $c$ is locked.

Algorithm 3.3 presents how the verification can be performed by considering all the pairs of routes for route-monotonic application data. The *command* and *prove* instructions (lines 5 and 7) correspond to the requests defined in the application data, such as in Listings 2.1 and 2.4. The bidirectional locking request (Listing 2.3) is also done through the *command* instruction. These instructions return *True* if the request is fulfilled and *False* otherwise. Furthermore, if they are accepted, all the attached actions modifying the station state are executed. The *move* instruction (lines 20 and 23) moves a train to the next track segment as defined by the state of the points. If a point is misplaced, the train will either derail or pursue its movements until it leaves the station. We will then consider that the position of the train is not included in the the set of all track segments anymore.

First, each pair of routes is considered (lines 1-2). The goal is to move a train $t_1$ from the origin of a route to its destination (lines 10-28). For each

**Algorithm 3.3:** No conflictual pair of routes for route-monotonic cases.

```
1  for r1 ∈ ROUTES do
2      for r2 ∈ ROUTES such that r2 ≠ r1 do
3          place a train t1 at r1.origin
4          place a train t2 at r2.origin
5          r1.isCommanded ← command(r1)
6          r2.isCommanded ← command(r2)
7          r1.isProved ← prove(r1)
8          if not r1.isProved then
9              return False
10         while t1.position ≠ r1.destination do
11             if not r2.isCommanded then
12                 r2.isCommanded ← command(r2)
13             if r2.isCommanded and not r2.isProved then
14                 for p ∈ POINTS such that t1.position = p do
15                     if p.state ≠ previous(p.state) then
16                         return False // derailment detected (Requirement 2)
17                 r2.isProved ← prove(r2)
18             if r2.isCommanded and r2.isProved then
19                 while t2.position ≠ r2.destination do
20                     move(t2)
21                     if t1.position = t2.position then
22                         return False // collision detected (Requirement 1)
23             move(t1)
24             if t1.position ∉ TRACK_SEGMENTS then
25                 return False // path inconsistency detected (Requirements 3 or 4)
26             remove t2 from S
27             for c ∈ COMPONENTS such that c.isLocked do
28                 release(c)
29         reinitialise(S) // components are unlocked and trains removed
30 return True
```

position of $t_1$, we try to command and to prove another route (lines 12 and 17). We also try to command $r_2$ directly after $r_1$ has been commanded (line 6). If $r_2$ is successfully commanded and proved (line 18), we move a train $t_2$ until it reaches the destination of the route (lines 19-22). When a particular position of $t_1$ has been tested, $t_1$ goes to its next track segment (line 23) and the interlocking will try to release all the locked components (lines 27-28).

Releasing conditions are described in the application data such as in Listing 2.5. Through the iterations on the positions of $t_1$, we memorise the fact that the other route, $r_2$, has been commanded or proved (lines 12 and 17). Indeed, because of the succession of release actions, the command and the proving can occur at different moments during the route life cycle. When a pair of routes has been entirely tested, the station is reinitialised (line 29) in order to have an empty station before testing the next pair. It is done through *reinitialise* instruction which releases all the locked components and removes all the trains of the station.

Let us notice that once $r_2$ has been successfully commanded and proved, $t_2$ moves until it reaches the destination of the route. Two cases must be considered. On the first hand, $t_1$ is still on the remaining path of $t_2$ and a collision will happen (line 22). Such a case is then generated and detected by the algorithm. On the other hand, $t_1$ is not on the remaining path of $t_2$ anymore and $t_2$ can continue to move till then end of its route without risk of collision. Besides, according to Property 3.7, no new command of $r_1$ can be granted because $t_1$ is still on its route. In this case, no issue can happen and it is then not required to test situations where $t_1$ is not on the remaining path of $t_2$. Except this reduction, all the other possibilities involving pairs of routes are generated.

Requirement 1 is tested after each movement of $t_2$ by testing that its position can never be the same as $t_1$ (lines 21-22). Requirement 2 is tested each time $r_2$ has been commanded. If the current position of $t_1$ is a point, the point cannot move after the command of $r_2$ (lines 14-16). It is done by comparing its state with its previous one through the operator *previous*. Requirements 3 and 4 are tested on lines 24 and 25. If $r_1$ cannot be proved (lines 8-9), we consider that we have an availability issue because no other route is already proved. The concept of availability is not yet presented as a requirement and is detailed in the next chapter.

Concerning the time complexity, each pair of routes must be tested, as well as the complete path of a route. We have thereby the theoretical bound $\mathcal{O}(r^2t)$ with $r$ the number of routes and $t$ the number of track segments.

However, Algorithm 3.3 is designed for application data that are route-monotonic and has to be slightly modified for the general case of monotonic-

complete application data. The only difference is that paths must be considered instead of routes. For each $fp \in FULL\_PATH$, we also define $fp.origin$ as the origin of the first route composing $fp$, $fp.destination$ as the destination of the last route of $fp$, $fp.isCommanded$ and $fp.isProved$ as boolean values indicating if all the routes inside $fp$ are commanded and proved. Finally, we define *fullCommand* and *fullProving* instructions with Algorithms 3.4 and 3.5.

**Algorithm 3.4:** Definition of *fullCommand*.

```
1  INPUT: fp ∈ FULL_PATH.
2  OUTPUT: True if all the routes composing fp can be commanded.
3          False otherwise.
4
5  Function fullCommand(fp):
6      for r ∈ fp do
7          r.isCommanded ← command(r)
8          if not r.isCommanded then
9              return False
10     return True
```

**Algorithm 3.5:** Definition of *fullProving*.

```
1  INPUT: fp ∈ FULL_PATH.
2  OUTPUT: True if all the routes composing fp can be proved.
3          False otherwise.
4
5  Function fullProving(fp):
6      for r ∈ fp do
7          r.isProved ← prove(r)
8          if not r.isProved then
9              return False
10     return True
```

A full path is commanded only if all of its composing routes are commanded and similarly for the proving. Let us also mention that the *command* and the *proving* actions are done sequentially from the first route of $fp$ to its last one. It reflects the practical behaviour of the interlocking. If a command or a proving has been refused, all the previous routes and the related

components are released. The verification of monotonic-complete application data can then directly be inferred from Algorithm 3.3 by considering full paths instead of routes, *fullCommand* instead of *command* and *fullProving* instead of *proving*. The time complexity can now be expressed in terms of the numbers of possible full paths, which give the theoretical bound $\mathcal{O}(fp^2t)$ with $fp$ the number of full paths and $t$ the number of track segments. In practice, $fp$ does not exceed two times the number of routes.

## 3.5 Experimental Results

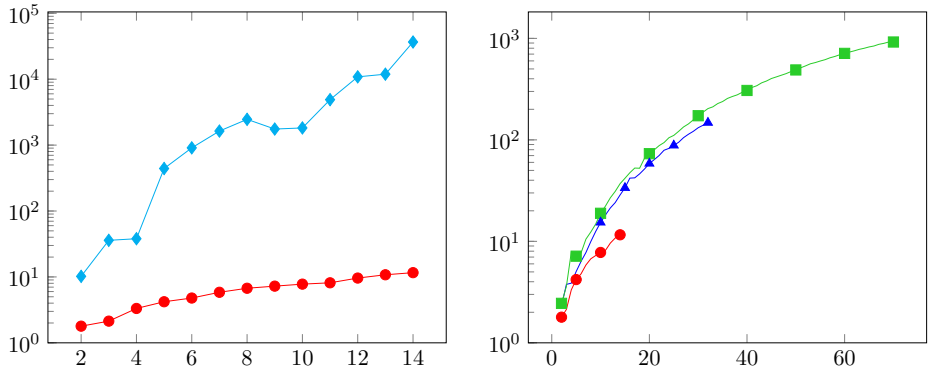
Several kinds of errors have been introduced in the application data in order to test the adequacy of the algorithm:

- A point is moved to a wrong position when a route is commanded (Appendix D.1).
- A subroute is not properly locked when a route is commanded (Appendix D.2).
- Some conditions are missing for releasing a component (Appendix D.3).
- Some conditions are missing for commanding a route (Appendix D.4).
- A bidirectional locking is not properly set when a route is commanded (Appendix D.6).

The other errors depicted in Appendix D are not related to safety, but to availability instead. All of the aforementioned errors have been successfully detected for the three case studies (Appendix B) by the algorithm in less than 30 minutes. Experiments have been realised on a MacBook Pro with a 2.6 GHz Intel Core i5 processor and with a RAM of 16 Go 1600 MHz DDR3 using a 64-Bit HotSpot(TM) JVM 1.8.

Besides, we compare our method with the approach of Busard et al. [BCL$^+$15] that have performed a model checking verification of Namêche without considering only pair of routes. Results are presented in Figure 3.2 (a). Let us notice that the y-scale is logarithmic. Execution time is presented in function of the number of routes considered. A complete verification requires to consider all the possible routes (14 for Namêche). Limiting the number of routes only produces a partial verification. As we can see, our

algorithm runs faster ($\approx$ 4 orders of magnitude for the complete verification of Namêche) than the approach where all the combinations of routes are tested. Similarly, Figure 3.2 (b) recaps the execution time for our three case studies. This speedup is mainly due to the restricted state space that we have considered. We can observe that the algorithm scales well for the three stations. Verification for Namêche, Braine l'Alleud and Courtrai takes around 20 seconds, 3 minutes and 16 minutes respectively.




(a) Comparison with [BCL$^+$15]. (b) The three case studies.

**Figure 3.2.** Execution time (in seconds) in function of the number of routes in Namêche (●), Braine l'Alleud (■) and Courtrai (▲) by using our algorithm and the model checking approach of Busard et al. [BCL$^+$15] for Namêche (⧫).

## 3.6 Future Work

Up to now, the verification proposed in this chapter is targeted for individual interlockings taken separately. However, it happens in practice that large stations are controlled by several interlocking systems. It is, for instance, the case of Courtrai, which is divided into three areas. New aspects must be considered, such as the communication between interlockings, which can share some variables. In addition to the correctness of each interlocking, the safety of the station is also dependent on the correctness of this communication.

As future work, we can extend the verification process in order to take this communication into account and then completely verify the safety of an entire station. We identified two possibilities to do so. On the first hand, we

can extend the concept of full paths in order to allow them to encompass routes which can be linked to different interlockings. The principles of the communication have then to be integrated in the algorithm. On the other hand, we can proceed in two steps. First, we verify each interlocking taken separately using model checking and the reduced state space. Then, we verify that the communication is also correct. It can be done using a compositional approach, as proposed by Limbrée et al. [LCPT16].

## 3.7 Summary

Much research has been carried out in order to automatically verify the correctness of an interlocking system. Up to now, most of it tackles the problem with a model checking approach which suffers from the state space explosion problem. In this chapter, another approach is proposed. The idea is to use our knowledge of the system not only for its modelling, but also for designing the verification algorithm. Concretely, we proposed a proof which can be used for reducing the state space, and we implemented a dedicated polynomial algorithm that can ensure the safety of a large station in less than one hour. The validity of this method is corroborated by the successful detection of errors that were introduced in the application data.

Finally, the method proposed here only deals with the verification of safety. Availability properties, stating that the trains will always progress in the station, are not considered. Whereas CENELEC Standard EN50128 strongly recommends the use of exhaustive methods for the verification of safety, the verification of availability can be based on non exhaustive methods as Statistical Model Checking. This aspect is the topic of the next chapter.

# Chapter 4
# Availability Verification

*"A ship is safe in harbor. But that is not what ships are for."*

–John A. Shedd

## 4.1 Motivation

Beyond the safety, a correct interlocking must also ensure that no train will be stopped too long in the station in order to maintain its *availability*. For this reason, availability requirements must also be considered. It ensures that every train will progress in the station without being locked because of improper interlocking operations. Availability of the traffic is dependant of the availability of the components. Components unnecessary locked for a train will prevent their usage for other ones. Ensuring the availability of a station is not trivial and requires to consider the different steps of route life cycles, which can be interleaved. A static analysis of the application data is then not sufficient.

The previous chapter presented a proof which can be used for easing the verification of safety requirements. However, this approach is based under the assumption that considering only pairs of full paths is sufficient for verifying the correctness of monotonic-complete application data. Unfortunately, this assumption does not hold when availability properties are considered. Indeed, an inconsistency can occur after a succession of interleaved operations involving more than two full paths. For instance, let us consider the fictive conditions for releasing Subroutes `U_1`, `U_2` and `U_3` as depicted in Listing 4.7. Let us also consider that each subroute is locked by a different full path. If the interlocking has allowed the three subroutes to be locked together, they can not be released thereafter and an availability issue occurs. This case would not be detected if only two full paths were considered.

```
1 U_1 f if U_2 f
2 U_2 f if U_3 f
3 U_3 f if U_1 f
```

**Listing 4.7.** Fictive example for releasing subroutes.

Until now, verification of availability requirements is still subject to the drawbacks of model checking and the state space explosion problem. Although CENELEC advocates the use of exhaustive methods for the verification of safety, availability requirements can be verified through a less restrictive process. It is the aim of this chapter. The objective is twofold: designing a method which is not limited by the state space explosion for practical uses but which nevertheless provides enough guarantees on the correctness of the system. The solution proposed is based on random simulations enriched with Statistical Model Checking. This approach is divided into different steps:

1. A model reflecting the behaviour of an interlocking is built from the combination of the application data and the track layout of the corresponding station.
2. The model is simulated several times and a set of traces is then obtained for *a posteriori* analysis.
3. The requirements that an interlocking must satisfy in order to ensure the availability of the station are formalised.
4. Statistical Model Checking algorithms are used in order to evaluate the probability that the interlocking model satisfies the availability requirements thanks to the outputs of the simulations.

Figure 4.1 presents a global overview of this approach. The different steps are detailed in the sections composing this chapter. Experimental results showing the adequacy and the scalability of the approach are then proposed. Finally, the last section shows how the approach can be used as a general framework for verifying properties related to the interlocking and that go beyond the scope of availability.

## 4.2 Interlocking Model

The first step is to build a model reflecting the behaviour of an interlocking. To do so, the model must rely on two data sources: the application data and

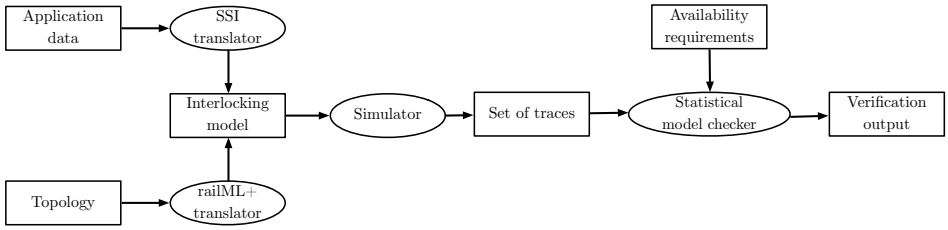



**Figure 4.1.** Overview of the approach for the verification of availability requirements.

the track layout. Although the application data describe fully the behaviour of an interlocking, they contain no information about the track layout of the considered area. However, the correctness of an interlocking is also dependent of its consistency with the track layout. It is why a data source describing the track layout is also required. As indicated in Section 2.5, railML+ is the format used.

The construction of the model is based on two translators that parse and aggregate both data sources into a single model. Parsing is carried out thanks to grammars which encapsulates the intrinsic structure of the input data. Grammars for application data expressed in SSI are formalised in Appendix C. They are translated using a combinatory parsing technique [Hut92] with the implementation of Moors et al. [MPO08]. On the other hand, the parsing of the track layout is carried out with the implementation proposed in the XML Scala library which is based on the standard SAX parser from Java library [MB02].

The architecture of the resulting model is presented in Figure 4.2. It is composed of five specific modules.

**Configuration** It encapsulates the interlocking behaviour as described in its application data and defines the logical components used.

**Station Topology** It represents the track layout of the considered area. To do so, graph theory [W+01] is used. This structure has already been deeply studied for modelling complex networks [Str01] such as Internet [FFF99], social networks [Bor12], transportation networks [TR95] or for the specific case of railway signalling [Won91]. In our case, the station is modelled as

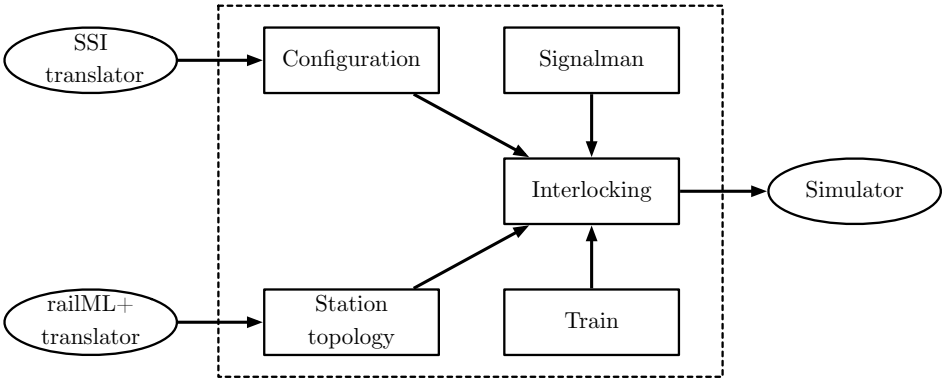



**Figure 4.2.** Architecture of the interlocking model.

an undirected graph. Physical components (signals, points and joints) are represented by nodes and their connections by edges. Each edge belongs to a particular track segment. The implementation is done with Jung [OFWB03]. The environment and special infrastructure (level crossing, tunnel, bridge, etc.) is not considered.

**Train** Each train follows routes in order to reach its destination. The train model is based on several assumptions:

- Trains properly follow the signalling principles. For instance, they do not overrun signals at a red aspect.
- Their speed and their length are abstracted.
- They can occupy one and only one track segment at each time step.
- They have no technical issue such as faulty components.

Such assumptions are also proposed by different authors [BCL+15, WJR+06, TRN+02].

**Signalman** In few words, the signalman is the person controlling the railway traffic by setting the signals and the points. For route based interlockings, signalmen have the responsibility to perform route requests to the interlocking when trains enter into a station. This behaviour is modelled by performing route commands (Listing 2.1) when a train is waiting at a start signal.

**Interlocking** It aggregates the previous models in order to have an executable representation of an interlocking which can be simulated thereafter. As suggested by Winter et al. [WJR+06], the model is designed to be as simple as possible. To do so, the modelling relies on several assumptions:

- Signals have only two aspects. They move from a stop aspect to a proceed aspect immediately after a route has been proved. The abstraction of other aspects (such as *can proceed at a restricted speed*) is a direct implication that speed of trains and the environment is abstracted.
- There is no distinction between a route and a shunting route.
- There is no component failure.
- Communication between interlockings are not considered.

## 4.3 Simulation of the Model

The next step is to simulate the model in order to obtain a representation of the interlocking behaviour. Simulation is a technique that has already been applied to a large number of fields such as weather forecasting [JBJ08], logistic [FFdS+15], healthcare facilities [WUS+10, JSL13] or transportation [RN07]. It has also been applied to the railway field. For instance, Sogin et al. [SBS11] analyse through simulations the effects of higher speed passenger trains in freight networks. Besides, OpenTrack provides a railway simulation tool [NH04] in order to predict some information about future situations of the traffic. Informally, a simulation can be defined as *an imitation of a system* [Rob14]. Its main asset is that it allows the study of various systems without building the system, thus saving precious time, cost and effort.

There exist several ways to carry out a simulation, this choice is highly dependent on the considered system and the desired analysis. Sulistio et al. [SYB04] have designed a taxonomy (Figure 4.3) encompassing the main choices to do when performing a simulation:

- **Presence of Randomness**: a simulation can be either *deterministic* or *probabilistic*. Deterministic simulations ensure that from a given input, the output will always be the same. It is not the case in probabilistic simulations where randomness is present.

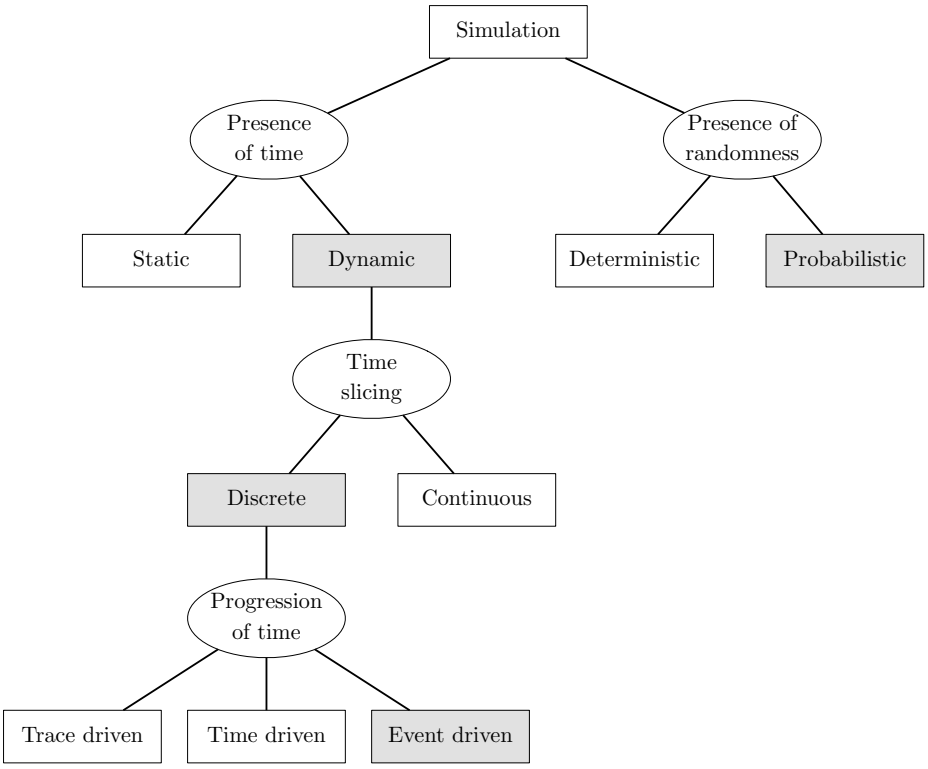



**Figure 4.3.** Simulation taxonomy with the characteristics of our simulation (in grey).

- **Presence of Time**: a simulation can be either *static* or *dynamic*. Static simulations imitate systems at a specific time while dynamic simulations also consider the progression of the system through time.
- **Time Slicing**: a dynamic simulation can be either *continuous* or *discrete*. On the one hand, continuous simulations consider an infinity of values for each time interval. Differential equations are often involved. On the other hand, discrete simulations split the time into a finite set of values called *instants*.
- **Progression of Time**: the time management in a discrete simulation can be done in three ways. Firstly, *trace driven simulations* progress by reading a set of events collected a priori from another environment. On the contrary, execution time of events can also be discovered during

the simulation itself. It this the case in *time driven simulations* where the progression is done by fixed time increments and with *event driven simulations* which progress with irregular time increments by jumping from events to events.

Most of computer simulations are dynamic and probabilistic. Furthermore, simulating a system through a discrete event simulation is often more suited for verification purposes. For such reasons, they are the characteristics we have chosen. The architecture of the simulator is presented in Figure 4.4. It is composed of four main modules.

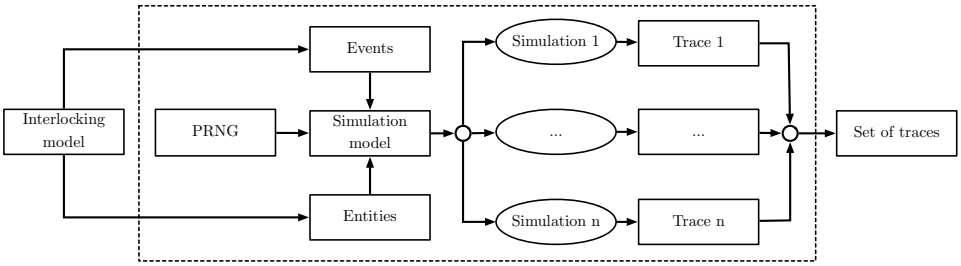



**Figure 4.4.** Architecture of the simulator.

**Entities** Informally, entities are the active objects that can be modified during a simulation. Each of them is characterised by a state composed of a list of attributes with mutable values. Two kinds of entities are involved:

1. **Interlocking Entities**: they are directly inferred from interlocking components such as the points, signals, routes, etc. All of them, as well as their possible values, have been presented in Table 2.1.
2. **Trains Entities**: they are obtained from the train model. They are characterised by two attributes: a *position*, which corresponds to the current track segment occupied by the train, and a *direction* (`up` or `down`). By convention, `up` direction goes from the left to the right (see Figure 2.2 for instance).

**Events** They define the actions that can change the entity states. Such actions can also generate new events. All of them have an execution time and can be guarded by conditions that must be satisfied before the event execution.

There are events related to the application data and events triggered by the train movements. This second category encompasses three events:

1. **Train Arrivals**: trains arrive randomly in the station at a start signal. Such an event can occur with a uniform probability in the interval $[t_a, t_a + \beta_a]$ where $t_a$ is the time of the last train arrival ($t_a = 0$ at initialisation) and $\beta_a$ is a predefined parameter. Besides, after each occurrence of this event, a new event is triggered in the interval $[t_a, t_a + \beta_a]$ with the updated $t_a$.
2. **Train Movements**: trains move through the station from track to track by following the direction set by the signals and the points. The first movement of a train $x$ is triggered when its route has been proved. The next movements occur in the interval $[t_m(x), t_m(x) + \beta_m]$ with $t_m(x)$ the time of the last movement done by $x$ and $\beta_m$ a parameter. Each train has thereby its own queue of events. It implicitly models the fact that the speed of trains can be different.
3. **Train Departures**: when a train reaches the end of its full path, it is removed from the station.

Besides, four other events can be inferred from the application data:

4. **Full Path Generation**: Route requests (as in Listing 2.1) for routes composing a full path for trains waiting at a start signal are periodically issued. A request is accepted only if all of its conditions are fulfilled. The route is then commanded and all the actions described in the request are executed. Otherwise, the request is aborted and no action is taken. Such an event can occur in the interval $[t_r, t_r + \beta_r]$ with $t_r$ the time of the last full path generation. Related conditions for moving a point (Listing 2.2) and for setting a bidirectional locking (Listing 2.3) are also considered during this event.
5. **Route Proving**: this event is periodically issued for routes waiting to be proved. If all the conditions are fulfilled, the route is proved. Otherwise, the request is discarded and no action is taken. This event is related to route proving requests, as in Listing 2.4.
6. **Route Destroying**: it performs a hard release of a route. The route is unset and each of its components is released. This event is triggered after three discarded *route destroying* events for a same route.

7. **Component Releasing**: it tries to release every locked component, as in Listing 2.5. This event is periodically issued after each train movement.

It gives a total of seven events. For the execution time of several events, randomness is introduced through the $\beta$ parameters. Their purpose is to allow the generation of a broad set of scenarios at each simulation by defining a time range on which the events can occur. To do so, the values have to be high enough for generating all the combinations of interleaved scenarios. The parameter $\beta$ for a particular event also determines the occurrence frequency of this event. A lower $\beta$ value for an event $e_1$ than for the others indicates that $e_1$ will have a higher probability to occur. In our simulation, each $\beta$ has the same value. It indicates that each event has the same probability to occur. It could also be done using a Poisson distribution with a constant rate. Figure 4.5 illustrates a possible scenario for Braine l'Alleud (Figure 2.2).

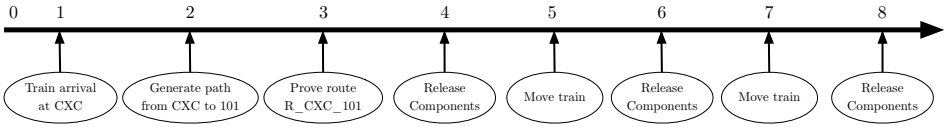



**Figure 4.5.** Possible scenario generated from the events.

The event list is implemented with a priority queue, ordered by event time. For that, a heap data structure [Cor09] is used.

**Pseudo Random Number Generator** As previously indicated, randomness is present in the simulation. However, generating pure random values with a computer is not feasible [Gen06]. A Pseudo Random Number Generator, or PRNG, can be used to tackle this problem. Informally, a PRNG is an algorithm which can generate from an initial value, a sequence of number which looks like a sequence of numbers randomly picked up. The initial value is also called the *seed*. For a given seed, a PRNG always produces the same sequence of numbers. There exists a plethora of PRNG in the literature [MN98, BBS86, WH82, EL86], each with their own specificities. Concerning the seed, it is often determined by unpredictable parameters having a high level of entropy such as the exact timing of keystrokes or the movements of the computer mouse [FSK10].

The PRNG used follows a linear congruential generator pattern [Knu98] with a 48-bit seed. It is based on the following recurrence relation:

$$X_{n+1} = (aX_n + b) \mod m$$

where $a$, $b$ and $m$ are parameters. The next number of the sequence if then determined from its previous number. We chose the implementation proposed in the standard library of Java (`java.util.Random`) which is suitable for generating numbers with a uniform probability. Although this PRNG is not cryptographically secure [Kel01], this property is not required for our simulation.

**Simulation Model** Obtained from the aggregation of the previous modules, this module gives a model which can be simulated. Concretely, it gathers information about the interlocking and defines how its behaviour can be simulated when facing up to the traffic. All the information about the current situation is gathered into a *simulation state*. A simulation state with $i \in \mathbb{N}$ is defined in Equation (4.1).

$$s_i : \left\langle nb, \sigma_p, \sigma_r, \sigma_s, \sigma_{uir}, \sigma_{ubsi}, \sigma_{track}, \sigma_{train} \right\rangle \tag{4.1}$$

In this equation, $s_i$ is the state at the $i^{th}$ step of the simulation. Let us state `point`, `route`, `subroute`,`uir`, `ubsi`, `track` as the sets of the interlocking components as defined in the application data and `train` as the set of trains in the station. The other variables are then defined as follows:

- $nb$ is the number of trains that have moved in the station so far. This information is used thereafter in order to under-approximate how many real days the simulation has covered.
- $\sigma_p$ : `point` $\rightarrow$ {`normal`, `reverse`, `default`} is a function defining the position of a point. The `default` state represents a point that is not positioned yet.
- $\sigma_r$ : `route` $\rightarrow$ {`unset`, `commanded`, `proved`} is a function defining the state of a route.
- $\sigma_s$ : `subroute` $\rightarrow$ {`free`, `locked`} is a function defining the state of a subroute.

- $\sigma_{uir}$ : `uir` $\rightarrow$ {`free`, `locked`} is a function defining the state of an immobilisation zone.
- $\sigma_{ubsi}$ : `ubsi` $\rightarrow$ {`free`, `locked`} is a function defining the state of a bidirectional locking.
- $\sigma_{track}$ : `track` $\rightarrow$ {`clear`, `occupied`} is a function defining the state of a track segment.
- $\sigma_{train}$ : `train` $\rightarrow$ (`track`, {`up`, `down`}) is a function defining the current position of a train and its direction.

Besides, the model also includes an event queue. Equation (4.2) defines a simulation model $sm$ in terms of a simulation state $s$ and an event queue $E$.

$$sm : \left\langle s, E \right\rangle \tag{4.2}$$

At each state, the first event of the queue is processed and then removed. Table 4.1 presents a subset of the simulation states corresponding to the scenario depicted in Figure 4.5 for Train `IC 442`. Symbol ✓indicates that the event associated to the transition has been accepted while ✗ indicates that it has been refused. For instance, no action is done during the transition from $s_3$ to $s_4$ because the event related to this transition (*component releasing* event) has been refused.

| | R_CXC_101 | P_01BC | P_02AC | UIR_(01BC) | IC 442 |
|---|---|---|---|---|---|
| $s_0$ - | unset | default | default | free | - |
| $s_1$ ✓ | unset | default | default | free | (up, T_092) |
| $s_2$ ✓ | commanded | normal | reverse | locked | (up, T_092) |
| $s_3$ ✓ | proved | normal | reverse | locked | (up, T_092) |
| $s_4$ ✗ | proved | normal | reverse | locked | (up, T_092) |
| $s_5$ ✓ | proved | normal | reverse | locked | (up, T_01BC) |
| $s_6$ ✗ | proved | normal | reverse | locked | (up, T_01BC) |
| $s_7$ ✓ | proved | normal | reverse | locked | (up, T_101) |
| $s_8$ ✓ | proved | normal | reverse | free | (up, T_101) |

**Table 4.1.** Simplified trace of the scenario presented in Figure 4.5. Symbol ✓indicates that the event associated to the transition of state has been accepted while ✗ indicates that it has been discarded.

The simulation stops either when the event queue is empty, or when an ending condition has been reached. We define the ending condition in function of the number of trains that have moved in the station. The simulation is stopped when $nb$ has reached a threshold $\Theta$. More details about the choice of this threshold are provided in Section 4.5. Algorithm 4.1 shows how the simulation is executed.

**Algorithm 4.1:** Interlocking simulation.

```
1  INPUT: A simulation model sm.
2  OUTPUT: A trace (sequence of states) of the simulation of sm.
3
4  trace ← sm.s_0
5  i ← 0
6  while sm.s_i.nb < Θ and not sm.E.isEmpty do
7      e ← sm.E.pop
8      i ← i + 1
9      e.process // update the simulation state
10     trace ← trace.append(sm.s_i)
11 return trace
```

It takes as input a simulation model and returns the resulting trace. While the ending condition is not satisfied or while the event queue is not empty (lines 6 to 10), an event is popped from the priority queue (line 7) and is then processed (line 9). Finally, the new state is appended to the trace (line 10). The whole simulation engine, as well as the different events, has been implemented using the discrete event simulation package of OscaR [Osc12], a Scala toolkit for solving operations research problems. This toolkit has similar features as SimPy [MV03]. Several simulations can be performed. Given their randomness, each of them can give a different trace.

## 4.4 Availability Requirements

The next step is to define the requirements that must be satisfied during the simulation. As previously indicated, availability refers to the ability to provide a traffic which progresses continuously. From the interlocking point of view, the requirements can be expressed in terms of its logical components.

**Requirement 5** (Route setting)**.** *Routes can always be eventually proved.*

**Requirement 6** (Component releasing)**.** *Components can always be eventually released. In other words, they do not remain locked forever.*

*Always eventually* means that after any state of the system, there exists at least another state satisfying the requirement. Unlike safety requirements that could be expressed by means of invariants, such logic is not sufficient to express availability requirements. It is why a *linear temporal logic* [HR04] is considered. In few words, such a logic is designed to encode properties involving a sequence of states. To do so, it introduces five temporal operators. Let us consider arbitrary properties $\varphi$ and $\psi$. The operators are then:

1. $\bigcirc\varphi$ (*next*), which indicates that $\varphi$ has to hold in the next state.
2. $\Box\varphi$ (*globally*), which indicates that $\varphi$ has to hold in all the next states.
3. $\Diamond\varphi$ (*finally*), which indicates that $\varphi$ has to hold in at least one of the next states.
4. $\varphi U\psi$ (*until*), which indicates that $\varphi$ has to hold until a state satisfying $\psi$ has been reached.
5. $\varphi R\psi$ (*release*), which indicates that $\psi$ has to hold unless $\varphi$ is reached including where $\psi$ is reached.

Furthermore, for practical uses, operators can be enriched with bounds defining on how many iterations the operator must be considered. Such a logic is often referred in the literature as a *bounded linear temporal logic* [Kam12]. Requirements 5 and 6 can then be formalised:

$$\Box_n\Diamond_n(r.isProved) \qquad \forall r \in ROUTES \quad (4.3)$$

$$\Box_n\Diamond_n(\neg c.isLocked) \qquad \forall c \in COMPONENTS \quad (4.4)$$

where $n$ is the operator bound. This combination of bounds indicates that an availability issue will be detected if a route is not proved after at most $n$ simulation steps. The same also apply for the components releasing. It produces a simulation of $2n$ steps. The combination of the temporal operators $\Box$ and $\Diamond$ in the property $\Box\Diamond\varphi$ describes the fact that a state satisfying $\varphi$ must be visited infinitely often. It is also referred as a *liveness* property.

## 4.5 Verification with Statistical Model Checking

Finally, the traces of the simulation can be analysed in order to evaluate the correctness of the interlocking. As for the verification of safety, classical model checking approaches do not scale for large stations because of the state space explosion problem. This limitation is especially true for the verification of availability properties that are expressed in a linear temporal logic. Besides, to the best of our knowledge, there exists no similar proof as in Section 3 which can be used for pruning the search space. Another method is then required.

Although the highest safety integrity level is required for the verification of safety requirements, availability can be verified through a less restrictive process. We propose first to carry out the verification through *random simulations*. The idea is to simulate the system many times and to observe if the requirements are always satisfied. If no issue has occurred in any simulation and provided that the simulation time was long enough, we can then have a high expectation that the system is correct. Besides, the detection of one issue directly implies that the system is incorrect. Compared to model checking where all the states are considered, the random simulations only consider situations that can potentially happen in practice. Despite its simplicity, this method raises three fundamental questions:

1. How long should the simulations last at least ?
2. How many simulations should be executed in order to have enough confidence on the model correctness ?
3. How can we be sure that conflicting scenarios have been generated through the simulations ?

Each of them must be answered in order to have a verification reliable enough for practical uses. To do so, Statistical Model Checking (SMC) is considered [LDB10]. The aim of SMC is to approximate, in a controlled manner, the probability of satisfaction or violation of a property. Unlike classical model checking approaches where an exhaustive exploration of the state space is conducted, only a sample of simulations is required. Such algorithms have already been used for verification of several applications [ABF+16], often stochastic [SVA05], as biological [JCL+09], biochemical [CFL+08], electronic [CDL10] or aircraft [BBB+10] systems. To the best

of our knowledge, it has never been used for the verification of railway interlocking systems yet. Let us now provide an answer to our three questions.

**Setting the Simulation Time** The first question is related to the simulation time: how can we be sure that a simulation is long enough for having a verification that fully cover the interlocking behaviour ? For instance, some scenarios involving a train will never be generated if the simulation always stops when the train is in the middle of a station. It is why the choice of the simulation time is crucial. According to the aforementioned example, the bound must be sufficient to determine a simulation time long enough to allow a verification covering at least a complete scenario.

A *complete scenario* is a scenario going from a train arrival in a station to its departure. The scenario is not complete if the verification is aborted when the train is still waiting or moving into the station. For instance, a verification covering one hour will not be sufficient because trains arriving in a station could still be in it after this time. In our case, we assigned a simulation time allowing verifications covering a complete day. This value is chosen under the reasonable assumption that a train would not stay into the same station longer than one day. With such a bound, we can have the certainty that all the situations have a non zero probability to occur during a simulation and that all the possible scenarios can be covered through the verification. Given the operator bounds of Requirements 4.3 and 4.4, a total simulation time of two days is required in order to allow verifications covering one day. Variable $nb$ in Equation 4.1 is used in order to under approximate how many real days the simulation has covered. Indeed, by taking the extreme case of a busy station where there is an incoming train every minute all the day long, we can safely assume that the simulation has covered at least one real day when 1440 trains have moved through the station. Taking this value as the bound can then ensure that the simulation will allow verifications covering at least a complete scenario. Table 4.2 recaps the execution time for computing a 2-day simulation on the three case studies (Appendix B) and the corresponding number of time steps with $\beta = 100$. As we can see, the larger is the station, the longer is the time required to perform the simulation. These experiments have been realised on a MacBook Pro with a 2.6 GHz Intel Core i5 processor and with a RAM of 16 Go 1600 MHz DDR3 using a 64-Bit HotSpot(TM) JVM 1.8.

| | Namêche | Braine l'Alleud | Courtrai |
|---|---|---|---|
| Exec. time | 47 seconds | 6 minutes | 8 minutes |
| # time steps | 200000 | 800000 | 900000 |

**Table 4.2.** Approximate execution time and the corresponding number of time steps required for computing and verifying a 2-day simulation on the three case studies with $\beta = 100$.

**Setting the Number of Simulations** The second question tackles the problem of the confidence that we can have on the verification process. Considering only a single simulation can be risky. Indeed, given the randomness present in the simulation, it is possible that the simulation becomes too specific and that it does not reflect the behaviour of the entire system with enough accuracy.

A better idea would be to execute several simulations and to observe if the verdict is the same. Intuitively, the greater the number of simulations performed, the better our expectation on the system correctness. Aggregation of several simulations can be done with *Monte Carlo* methods [GS05, RK11]. It is used to estimate a probability $\gamma$ of satisfying a property $\varphi$ such as in Equations (4.3) and (4.4). The principle is to generate $N$ random simulations $\rho_1, \ldots, \rho_N$ and to compute an estimation of $\gamma$. Equation (4.5) illustrates this estimation.

$$\tilde{\gamma} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\rho_i \models \varphi) \tag{4.5}$$

The term $\mathbf{1}$ is an indicator function that returns 1 if $\varphi$ is satisfied in $\rho$ and 0 otherwise. The next step is then to determine $N$, the number of random simulations. To do so, *Chernoff bound* [Che52] is the method commonly used. It determines the number of simulations required in order to have a confidence $\delta$ and a precision $\epsilon$ on the probability. This relation is expressed in Equation (4.6).

$$Pr(|\gamma - \tilde{\gamma}| < \epsilon) \geq 1 - \delta \quad \text{if} \quad N \geq \frac{ln(\frac{2}{\delta})}{2\epsilon^2} \tag{4.6}$$

This bound $N$ guarantees that the probability that a property is satisfied is included in the $(1-\delta)$-$[\gamma - \epsilon, \gamma + \epsilon]$ confidence interval assuming that the

simulation is distributed uniformly with regard to real executions. In our case, we are interested in $\tilde{\gamma} = 1$ which gives the interval $(1-\delta)$-$[1-\epsilon, 1]$ It can then be used in order to have a parametrizable confidence on the correctness of the system. For instance, a 0.99-$[0.99, 1]$ ($\epsilon = 0.01$ and $\delta = 0.01$) can be obtained with 26 492 simulations of two days each. Given the bounds $\Box_n \Diamond_n$ of Requirements 5 and 6, each simulation embeds $n$ verifications covering one day of interlocking operations. Furthermore, refining $\delta$ by a factor 10 will increase the number of simulations by 11513 while refining $\epsilon$ by the same factor will multiply this number by 100. Knowing the execution time required to perform a 2-day simulation for each station from Table 4.2, we can deduce the expected execution time required to obtain such confidence intervals. A recap of the values obtained is presented in Table 4.3.

| | Namêche | Braine l'Alleud | Courtrai |
|---|---|---|---|
| Number of simulations | 26 492 | | |
| Real time covered | 144 years | | |
| Total exec. time | 14 days | 110 days | 148 days |
| Added exec. time for $\frac{\delta}{10}$ | +6 days | +48 days | +64 days |
| Added exec. time for $\frac{\epsilon}{10}$ | $\times 100$ | | |

**Table 4.3.** Characteristics of the verification for a 0.99-$[0.99, 1]$ confidence interval using Chernoff bound.

These results show that even poor confidence intervals (0.99-$[0.99, 1]$) for a small station require a consequent execution time. In other words, the limitation is the same as with the model checking approach: a high guarantee of correctness requires the execution of a huge number of simulations and then the exploration of too many states. In its simplest form, this method cannot be used in practice.

However, a crucial advantage compared to model checking is that random simulations can be easily multi-threaded without any overhead. By Gustafson's law [Gus88], the execution of $N$ simulations on $m$ processors will be $m$ times faster than the execution of $N$ simulations on a single processor. Even if model checking can also be paralleled [BBR07, Hol06], it is a harder task and the parallelisation gain decreases with the number of processes.

Table 4.4 summarises the number of processors required in order to obtain

a (0.99-[0.99, 1]) confidence interval on the verification of the three case studies. These results show that it is possible to reach strong Chernoff bounds in practice thanks to parallelisation. For comparison, the EC2 cloud computing services proposed by Amazon contain more than 30 000 cores [OIY+09]. Chernoff bound can then be used in order to achieve a parametrisable level of confidence on the correctness of the system.

| | Namêche | Braine l'Alleud | Courtrai |
|---|---|---|---|
| 1 hour | 336 | 2640 | 3600 |
| 12 hours | 28 | 220 | 300 |
| 1 day | 14 | 110 | 150 |
| 10 days | 2 | 11 | 15 |

**Table 4.4.** Number of processors required for performing the verification with a 0.99-[0.99, 1] confidence interval for different time periods.

**Verification Algorithm** Let us now describe how Requirements 5 and 6 are verified using a simulation trace. The idea is to check that each route would not stay unproved for more than one complete day. Similarly, components cannot be locked for more than one complete day. Let $s = \langle s_i, \ldots, s_m \rangle$ be the sequence of states obtained with a simulation of 2-days and $n$ the bound used for the availability requirements. The corresponding simulation trace will then contain $2n$ states. Algorithm 4.2 presents how the verification is performed.

First, a counter is assigned at each route and component and is initialised at $n$ (lines 6-7). Each simulation state is processed sequentially (lines 8-22). For each route, the algorithm checks if the route is proved at the current state (line 10). If it is not the case, the counter is decremented (line 11) and if it reaches zero, *False* is returned (lines 12-13). It means that the current route has not been proved during a complete day. Requirement 5 is then not satisfied. Once a route has been proved, its counter is reinitialised (lines 14-15). The same process is performed for the components (lines 16-22). The algorithm runs with a time complexity of $\mathcal{O}(n \times (r + c))$ with $n$ the simulation bound, $r$ the number of routes in the station and $c$ the number of components. However, it can also be directly embedded in the simulation and no post-processing of the trace is then required.

**Algorithm 4.2:** Verification algorithm based on the analysis of the simulation trace.

```
1  INPUT: A trace of m states ⟨s_i, ..., s_m⟩.
2         n, the verification bound (as in Requirements 5 and 6).
3  OUTPUT: True if both requirements are satisfied.
4          False otherwise.
5
   // initialisation of counters
6  routeMap[r] ← n                                          ∀r ∈ ROUTES
7  compMap[c] ← n                                           ∀c ∈ COMPONENTS
   // analysing each state of the simulation
8  for i ∈ 0, ..., m do
       // analysing the state of each route at s_i
9      for r ∈ ROUTES do
10         if not s_i.r.isProved then
11             routeMap[r] ← routeMap[r] − 1
12             if routeMap[r] = 0 then
13                 return False // r not proved for n steps: Req. 5 not satisfied
14         else
15             routeMap[r] ← n // reinitialisation of the counter
       // analysing the state of each component at s_i
16     for c ∈ COMPONENTS do
17         if s_i.c.isLocked then
18             compMap[c] ← compMap[c] − 1
19             if compMap[c] = 0 then
20                 return False // c locked for n steps: Req. 6 not satisfied
21         else
22             compMap[c] ← n
23 return True
```

**Generation of Conflicting Scenarios** A drawback of the random simulations is that they do not provide guarantees that particular scenarios have been generated. Therefore, it is theoretically possible that there exist conflictual scenarios not covered by the simulations. It is the topic of the third question. Similarly to code coverage [AO08] that is used for software testing in order to gain confidence into the quality of test suites, measures and statistical reports related to simulations can also be performed. Such measures can be for instance the number of times that a route has been commanded, proved, released and destroyed.

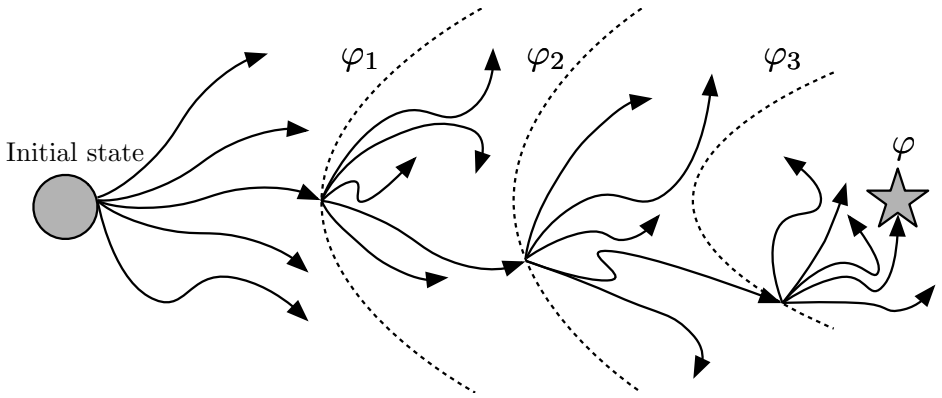



**Figure 4.6.** Importance splitting using three levels.

Other techniques for the generation of scenarios, particularly for the detection of rare events, such as importance splitting [JLS13, JLS14] or importance sampling [Sha94, Hei95] can also be used. Amongst them, importance splitting is particularly well suited for our application. It allows to increase the probability of generating rare events and to speed up the error detection by decreasing the number of simulations required to estimate the probability. It starts by splitting the property in a sequence of temporal properties $\varphi_k$, with the logical characteristic $\varphi = \varphi_M \Rightarrow \varphi_{M-1} \Rightarrow \cdots \Rightarrow \varphi_1$. It defines a set of levels, each associated to the conditional probability $Pr(\rho \models \varphi_{k+1} \mid \rho \models \varphi_k)$ of reaching level $k+1$ from level $k$. Instead of trying to detect directly a violation that occurs rarely, the importance splitting algorithm considers a set of sub-properties easier to verify and which lead progressively to the violation. An illustration of this process is presented in Figure 4.6.

However, the results presented in the next section shows that resorting to such a method is not required for availability purposes because of the good results obtained with simple statistical reports. It can thereby be useful when other properties, related to rare events, must be verified.

## 4.6 Experimental Results

The last step is to analyse the validity of the approach through experimental results. Indeed, now that we have a model and a verification process, we

need to ensure that it will efficiently detect the errors leading to availability issues. To do so, we introduced several errors in the application data in order to test if they will be detected through the verification. Introduced errors related to availability are as follows:

- Point moved to a wrong position when commanding a route (Appendix D.1).
- Irrelevant additional conditions for releasing a component (Appendix D.5).
- No consistency between a route command and a route proving (Appendix D.7).

These errors are known to cause availability issues. Other errors depicted in Appendix D are only related to safety. The simulator and the verification algorithm has been implemented in Scala. the Discrete Event Simulation package of OscaR [Osc12] has also been used for the simulator. Table 4.5 recaps the number of tests where an issue violating the availability requirements when errors are introduced. A hundred simulations have been performed for each configuration.

**Table 4.5.** Number of simulations, among 100, where an issue violating an availability requirement (Equations (4.3) and (4.4)) has been detected for the three case studies.

| Error | Namêche | | Braine l'Alleud | | Courtrai | |
|---|---|---|---|---|---|---|
| | (4.3) | (4.4) | (4.3) | (4.4) | (4.3) | (4.4) |
| D.1 | 100 | 0 | 100 | 0 | 100 | 100 |
| D.5 | 100 | 100 | 100 | 100 | 100 | 100 |
| D.7 | 100 | 0 | 100 | 0 | 100 | 100 |

A non zero value indicates that an availability issue has occurred on at least one simulation. In this case, the system is not correct. As we can see, each error introduced in the application data causes the violation of at least one requirement. Furthermore, we can also deduce that even if precise confidence intervals require a consequent execution time, errors are detected faster in practice.

## 4.7 Future Work

Possibilities of future work are multiple. Until now, the approach presented in this chapter has only been considered for the availability verification. However, it is not its only asset. The approach can be seen as a general framework for verifying properties related to interlocking systems. As depicted in Figure 4.2, new properties can be easily considered without modifying the model and the simulator. Verification of other properties can then be easily integrated.

For instance, let us come back on the verification of safety presented in the previous chapter. We have shown that a dedicated algorithm can be used in order to detect errors in the application data causing safety issues. However, the identification of which part of the application data is erroneous remains not trivial. This kind of problematic has been considered by Busard et al. [BCL+15] that identified a set of properties that the application data must satisfy in order to avoid safety issues. Such properties are used to detect what are the errors in the application data leading to an unsafe behaviour of the interlocking. Some of them, with examples related to Braine l'Alleud (Figure 2.2), are as follows:

- Routes in opposite directions (e.g. `R_CXM_101` and `R_DC_92`) cannot be proved at the same time.
- Subroutes in opposite directions (e.g. `U_CC_DXC` and `U_DXC_CC`) cannot be locked at the same time.
- Subroutes composing a route must be released in correct order. For instance, `R_KC_102` is composed sequentially by `U_KC_19C`, `U_19C_20C` and `U_20C_CGC`. The three subroutes must then be released in this order.
- A point cannot move when its attached immobilisation zone is locked. For instance, `U_IR(09C)` prevents `P_08AC` and `P_09C` to be commanded.

Identifying which properties are violated once a safety issue has been detected can give more information about which part of the application data is faulty. Verification of such properties can be directly performed without requiring any change in the model. When many properties are considered, it can be tedious to adapt the verification algorithm or to design a new one each time a property is added. A statistical model checker, such as

PLASMA Lab platform [BCLS13, JLS12], taking as input properties can be used instead. PLASMA Lab is a compact, efficient and flexible platform for verifying stochastic models with SMC. It includes several SMC algorithms and a library to include new simulators and to define properties. Simulators of systems or models can be reused with few implementation work thanks to the existing libraries. It already includes simulators for Reactive Module Language (Markov chains models as in the PRISM model-checker [HKNP06]), Simulink [Kar06] and SystemC [BDBK09] models. In our case, we have developed a new plug-in that implements PLASMA Lab library and creates an interface between PLASMA Lab and the simulator. Defining the new properties is part of the future work.

Besides, the model, and the simulator as well, can also be improved. As presented beforehand, they are built upon some hypothesis, which can be progressively removed. It will then be possible to model less usual behaviours (shunting, level crossing, etc.), to add characteristics to trains (speed, length, weight, etc.), to take into account the environment (tunnels, bridges, etc.), or to consider other aspects of signals (*can proceed at a restricted speed*). The simulation will thus be more realistic and the verification of other properties, not necessarily related to interlocking systems, become feasible. It is for instance the case of loading gauge verification, which defines the maximum dimension for trains to ensure safe passage through bridges, tunnels and other structures.

Probabilistic notions and rare events that can happen in practice can also be integrated in the simulation. For instance, the fact that a component failure will happen every ten years or that a driver will not follow the signalling principles every five year can be modelled. In this case, we are also interested in finding scenarios revealing a rare failure of the system and in inferring the probability that a safety or availability issue will occur under these failures. A knowledge of the system can be used in order to force the simulation to reach a failure, and to stop it if there is enough evidence that it will not reach a failure. Importance splitting is particularly well fitted for that.

Finally, other tools and methods can be used for modelling the system and its verification. For instance, *timed automata* [AD94] are particularly fitted for the verification of real-time systems. Methods for checking both safety and liveness properties in timed automata have been developed and

intensively studied over the last decades [BY04]. Several tools such as Kronos [BDM+98] or UPPAAL [LPY97] already exist.

## 4.8 Summary

The objective pursued in this chapter was to design a verification method which is not limited by the state space explosion for practical uses but which nevertheless provides enough guarantees on the correctness of the system. The key idea of the contribution is to perform several simulations and to observe through statistical tests whether the results obtained provide a statistical evidence that the system is correct with regard to availability. Requirements have been formulated, and experimental results have shown that the verification thoroughly detects violations of the properties. Besides, strong confidence intervals on the verification can be obtained thanks to parallelisation. We have finally shown that the approach proposed can be considered as a general framework for verifying properties related to interlocking systems.

# Chapter 5
# Fluidity Maximisation

*"All we have to decide is what to do with the time that is given us."*

– J.R.R. Tolkien, *The Fellowship of the Ring*

## 5.1 Motivation

Previous chapters presented how an interlocking can be verified in order to ensure the safety and the availability of the train traffic. However, although these requirements are necessary for commissioning an interlocking, they are not sufficient for providing a fluid traffic. Fluidity is often assimilated to availability but it is, in fact, a different notion. While availability is used to ensure that no train is blocked for too long, fluidity is related to the proper use of the railway resources such as the track segments. For instance, let us consider again the station of Braine l'Alleud (Figure 2.2). If Track `102` is never used whereas Track `101` is congested, the fluidity of the traffic inside Braine l'Alleud can then certainly be improved. Once the safety and the availability are ensured, the main goal pursued is then to maximise the fluidity. It is the problem addressed by this chapter.

While safety and availability issues are related to the interlocking, fluidity is beyond its scope and is dependent of other aspects of railway management. The first step is the development of a timetable. Over the years, the number of trains, the number of tracks and the complexity of networks keep increasing. In this context, the need of an efficient and reliable train schedule is crucial. Indeed, a bad schedule can cause train conflicts, unnecessary delays, financial losses, and a decrease in passenger satisfaction. Beyond the departure time, a schedule must also indicate on which platform trains must stop, and which itinerary they have to follow. While earliest train schedules could be built

manually without resorting to optimisation or computer based methods, it is not possible anymore.

The design of a proper and realistic timetable is then a necessary condition for preserving a fluid train traffic. A tremendous amount of research papers deal with this problematic of building the most appropriate schedule for general [HKF96, GSA04, ZZ05], or specific purposes [HS16, SWDK15]. This aspect is currently well achieved and its improvement is not a main concern. However, practical schedules must also deal with real time perturbations. Disturbances, technical failures or simply consequences of a too optimistic theoretical schedule can cause delays which can be propagated on other trains. The initial schedule may then become infeasible or suboptimal. Real-time modifications of the initial schedule may therefore be required. Dealing with such modifications is still a challenge today.

In Belgium, the responsibility of regulating the traffic is mainly performed by a software, called the Traffic Management System (TMS), that automatically manages the traffic according to predefined rules. According to an established timetable, this system commands routes in order to ensure the planned traffic. However, in case of real time perturbations, signalling must be handled manually and the actions to perform are decided by human operators. The general procedure follows this pattern:

1. The TMS has predicted a future conflict caused by perturbations on the traffic.
2. The operators controlling the traffic analyse the situation and evaluate the possible actions to do in order to minimise the consequences of the perturbations. Besides the safety and the availability, the criterion considered for the decision is the sum of delays per passenger. In other words, the objective is to minimise the sum of delays of trains weighted by their number of passengers. Furthermore, some trains can have a higher priority than others. For instance, connecting trains can have priority over other trains.
3. According to the situation, they perform some actions (stopping a train, changing its route, etc.) or do nothing.

When facing up real time perturbations, operators have to deal with several difficulties:

- The available time for analysing the situation and taking a decision can be very short according to the criticality of the situation (sometimes less than one minute).
- For large stations with a dense traffic, particularly during the peak hours, the effects of an action are often difficult to predict.
- The number of parameters that must be considered (type of trains, number of passengers, etc.) complicates the decision.

Such reasons can lead railway operators to take decisions that will not improve the situation, or worse, will degrade it. Most of the time, the decision taken is to give the priority either to the first train arriving at a signal, or to the first one that must leave the station [DPP07]. However, it is not always the best decision. For instance, let us consider the scenario depicted in Figure 5.1, where $t_1$ is a train that is 3 minutes away from the station, and $t_2$ another train, which is 10 minutes away. Both of them has to reach Platform $A$ but $t_2$ is already late and $t_1$ is ahead of schedule. Giving the priority to the first train arriving will allow $t_1$ to go first. However, in this case, stopping $t_1$ at the next signal and letting $t_2$ entering first in the station may be a better solution for reducing the total delay.

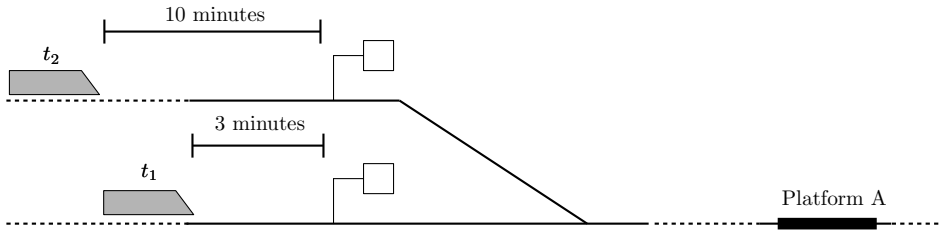



**Figure 5.1.** Situation where a decision is required in order to minimise the delay.

In this chapter, we propose a new method for rescheduling traffic in a real time context. It is divided into four sections:

1. The state of the art methods for tackling this problem are depicted.
2. The technical background on which relies our contribution is presented.
3. The model for rescheduling the traffic through real time disturbances on the railway network is fully described.

4. Experiments are carried out in order to corroborate the adequacy and the scalability of the approach. Concretely, its application on several situations in comparison with classical strategies of operators is performed.

## 5.2 Related Work

Aforementioned issues concerning real-time management of the traffic advocate the use of a decision support tool in order to assist operators in their decisions. The requirement for this software is then to provide a good solution to this rescheduling problem within a short and parametrisable computation time.

A recent survey (2014) initiated by Cacchiani et al. [CHK+14] recaps the different trends on models and algorithms for real-time railway disturbance management. For instance, Fay et al. [Fay00] propose an expert system using fuzzy rules and Petri nets for the modelling. Such a method requires to define the rules, which can differ according to the station. Higgins et al. [HKF97] propose to use local search methods in order to solve conflicts. However, this work does not take into account the alternative routes that trains can have in order to reach their destination, and that the planned route remains not always optimal in case of real time perturbations. For that, D'Ariano et al. [DPP07] model the problem with an alternative graph formulation and propose a branch and bound algorithm to solve it. They suggest to enrich their method with a local search algorithm for rerouting optimization purposes [DCPP08]. Besides, some works consider the passenger satisfaction [SS10, DHSS12, DHK+14] in their model. It is also referred as the *Delay Management Problem*.

Generally speaking, most of the methods dealing with railway traffic scheduling are based on Mixed Integer Programming (MIP) [CGD09, FLM+14, S. 83, LM15]. Informally, MIP is an optimisation technique where decision variables can be either discrete or continuous. However, the efficiency of MIP models for solving scheduling problems is known to be highly dependant of the chosen granularity of time. Furthermore, it is not as flexible and expressive for the modelling as other approaches such as Constraint Programming (CP).

Constraint Based Scheduling (CBS) [BLPN12], or in other words, applying CP [RVBW06] on scheduling problems seems to be a good alternative over MIP. According to the survey of Bartak et al. [BSR10], CP is particularly well suited for real-life scheduling applications. Furthermore, several works [KBK+12, KTK13, KB14] show that CP can be used for solving scheduling problems on large and realistic instances. By following this trend, Rodriguez [Rod07] proposes a CP model for real-time train scheduling at junctions. However, despite the good performances obtained, this model can be improved. Firstly, the modelling does not use the strength of global constraints which can provide a better propagation. Secondly, the search can also be improved with heuristics and the use of *local search* techniques. Finally, the objective function is only defined in function of train delays without considering the passengers or the different categories of trains. In this context, our contribution aims to tackle these flaws through an innovative way of modelling the problem.

## 5.3 Technical Background

Let us first introduce the technical background and all the information required for the comprehension of the subsequent chapters.

### Constraint Programming

Some scientists consider that CP is the holy grail of programming: the user states the problem and the computer solves it [Fre97]. In few words, CP is a general framework proposing simple, general and efficient algorithmic solutions to hard combinatorial problems. Two kinds of problem are targeted by CP. On the first hand, there is the Constraint Satisfaction Problem (CSP) where the goal is to find only a single feasible solution. Each solution satisfying the constraints is suitable. On the other hand, there is the Constraint Optimisation Problem (COP) where the goal is to find the best feasible solution according to an objective function. In order to solve both problems, two stages are considered. Firstly, the problem must be formally and declaratively represented by means of variables, constraints, and possibly objective functions. It is the *modelling* phase. Secondly, the modelled problem is solved by a tool which automatically search feasible solutions by exploring a search tree. It is the *search* phase. Let us describe in more details both phases.

### Modelling Phase

A CSP is defined by a set of *variables*, a set of *domains*, which contain possible values for the variables, and a set of *constraints*, which restrict assignments of values to variables. Formally, a CSP is defined by the triplet $(X, D, C)$ where:

- $X = \{x_1, \ldots, x_n\}$ is the set of variables.
- $D = \{D_1, \ldots, D_n\}$ is the set of the domain for each variable.
- $C = \{C_1, \ldots, C_m\}$ is the set of constraints.

The goal of a CSP is then to choose a value for each variable of $X$ from $D$ which satisfies all the constraints of $C$. A COP is an extension of a CSP where one, or several, objective functions are added. Formally, a COP is defined as the quadruplet $(X, D, C, O)$ where $O = \{o_1, \ldots, o_p\}$ is the set of objective functions. In addition to finding a feasible solution, the goal is to find the best assignation of variables according to the objective functions defined in $O$. Modelling a CSP, or similarly a COP, resorts then to define adequately the different sets.

### Search Phase

Once the problem is modelled, the next step is to solve it. To do so, two operations are interleaved: the *propagation* and the *search*. The search essentially consists of an exhaustive enumeration of all the possible assignments of values to variables until a feasible solution has been found. It is also possible that the considered problem has no solution or that we want to continue the search until the best solution has been found. A search tree, growing up exponentially with the number of variables, is constructed during this process.

Besides, propagations of constraints are applied in order to prune the search tree. The goal is to reduce the number of possible combinations. Given the current domains of the variables and a particular constraint $c$, the propagation of $c$ removes from the domains the values that cannot be part of the solution because of the violation of $c$. This process is repeated at each domain change and for each constraint until no more value can be removed from the domains. It is referred as the *fix point algorithm* (Algorithm 5.1). When the propagation has led to an empty domain for a variable, an

inconsistency is detected (line 4) which means that the current domains cannot give a feasible solution.

**Algorithm 5.1:** Fix point algorithm.

```
1 repeat
2 |   select a constraint c in C
3 |   propagate c
4 |   if an inconsistency is detected then
5 |   |   return False
6 until no value can be removed
7 return True
```

Concerning COP, the search process is improved with a *branch and bound* method [LW66]. When a feasible solution has been found, a new constraint ensuring that the next solution has to be better than the current one is added. Provided that the whole search space has been explored, the last solution found is then proved to be optimal.

## Constraint Based Scheduling

Scheduling [Pin15] is a generic term covering a large scope of optimisation problems related to time management such as *Job Shop Scheduling* [YN97], *Vehicle Routing Problem* [Lap92] or *Nurse Rostering Problem* [BDCBVL04]. In a nutshell, scheduling problems address two questions: what are the actions to do and when to do them ? Despite the simplicity of this formulation, most of the scheduling problems are computationally hard to solve. Solving big and realistic scheduling problems is still a challenge nowadays and is targeted by much research.

Among it, Constraint Based Scheduling [BLPN12] consists in applying CP for solving scheduling problem. The idea is to model the problem as a COP and to use efficient propagators [Vil07, Vil04, DVCS15, GHS15b, GHS15a, VCDMS16] in order to reduce the state space. According to the survey of Bartak et al. [BSR10], this approach seems to be particularly well suited for this task.

A scheduling problem is modelled by means of *activities*. Roughly speaking, an activity is an event that can be executed at some points in time. Each activity encapsulates three CP variables:

1. A *start date*, which defines when the activity can begin.
2. A *duration*, which defines the duration of the activity.
3. An *end date*, which defines when the activity can end.

These variables are linked by an implicit constraint of consistency stating that the end date of each activity must equal its start date in addition to its duration. Each activity is also characterised by a set of features. Two of them are considered here:

- An activity is *preemptive* if it can be interrupted and then restarted thereafter. It is *non-preemptive* if it cannot be interrupted.
- An activity is *optional* if it is not mandatory to be executed. It is *non-optional* if it must be executed.

Furthermore, activities are subject to a set of constraints. Constraints are the essence of scheduling problems. They define what we can do and when we can do it. Three categories of constraints are often considered.

Firstly, the *time-related constraints* encompass all the constraints restricting the time when activities can be scheduled. For instance, a *precedence constraint* among two activity states the second one cannot begin until the first one has been finished.

The second category contains the *resource-related constraints*. A *resource* is something that is needed by an activity during its execution. These constraints are used in order to express relations between activities and resources. For instance, the *unary resource constraint* [Vil04] prevents a resource to be used by more than one activity at each time.

Finally, a scheduling problem can also be defined by other constraints that are more general. For instance, the execution of two activities can be incompatible for some reasons that are not related to time or the resource usage. Solving a scheduling problem consists in finding a feasible assignment of the activities that satisfies all the constraints. However, in most cases, and especially in scheduling, finding a feasible solution is not sufficient. It is the best solution that is desired instead. To do so, an objective function has to be considered.

There is a large panel of possible objective functions for scheduling problems. For instance, we can chose to minimise either the *makespan*, or the *sum of delays*. The makespan is defined as the maximal ending time of the set of all activities in the schedule. When minimising the makespan, the goal is then to schedule the activities in order to finish the last one as soon as possible. Furthermore, each activity has commonly a *due date*, or in other words, a time defining when the activity should end. When an activity finishes after its due date, a *delay* is generated. The second objective function aims to minimise the cumulated sum of delays of all the activities.

### Time-Interval Variables

Many scheduling problems consider activities that may or may not be executed in the final schedule. As previously stated, such activities are optional. The choice of executing optional activities is then a decision that has to be done. Modelling and solving such activities is a hot topic in CBS. Most of the approaches are based on the definition of additional variables representing the existence of an activity in the schedule and the addition of new global constraints with dedicated propagators [BF99, MPP05, BC07]. Although an efficient propagation is done, the implementation of new constraints, only for optional activities, is required, which can sometimes be tedious.

Another approach is to design a dedicated structure which implicitly embeds the optionality nature of activities. It is the idea of the *time-interval variables* introduced by Laborie et al. [LR08, LRSV09]. Instead of defining new constraints over the additional variables in order to handle optional activities, the idea is to introduce new variables encapsulating the notion of optionality and to adapt the existing constraints to these new variables. Such an approach offers several advantages. For instance, it provides an expressive an easy modelling where the notion of optionality is implicitly contained in the new variables. No additional variable is required. Furthermore, an efficient propagation can be performed thanks to relations that hold between the variables.

In [LR08], Laborie and Rogerie have formalised the time-interval variables. A time-interval variable $a$ is a variable whose domain $dom(a)$ is a subset of $\{\perp\} \cup \{[s, e) | s, e \in \mathbb{Z}, s \leq e\}$. A time-interval variable is said to be *fixed* if its domain is reduced to a singleton. If $\underline{a}$ denotes a fixed time-interval, we

say that the variable is *non-executed* if $\underline{a} = \bot$ or that is *executed* if $\underline{a} = [s, e)$. Furthermore, we denote $s(\underline{a})$, or simply $s$, $d(\underline{a})$, or $d$, $e(\underline{a})$, or $e$, as the start date, the duration and the end date of $\underline{a}$ respectively. We also define the execution status $x(\underline{a})$, or $x$, which indicates if $a$ is executed or not.

The main characteristic of a time-interval variable is the way by which the constraints are managed. Any constraint involving at least one non-executed time-interval variable is not considered. In other words, such constraints have no impact on the model and no solution is removed through their propagation. Management of constraints is then hidden for the modelling and is implicitly done by the propagator and the solver engine. Let us introduce four kinds of constraints related to time-interval variables that are used in our model.

**Temporal Constraints** These constraints [LR08] express temporal relation between two activities. They are related to Allen's Relations [All83]. Their semantics depends on the temporal relation between the activities. The different possibilities are given in Table 5.1 for two fixed activities $\underline{a}$ and $\underline{b}$.

| Constraint | Semantics |
|---|---|
| `startBeforeStart`$(\underline{a}, \underline{b})$ | $x(\underline{a}) \wedge x(\underline{b}) \implies s(\underline{a}) \leq s(\underline{b})$ |
| `startBeforeEnd`$(\underline{a}, \underline{b})$ | $x(\underline{a}) \wedge x(\underline{b}) \implies s(\underline{a}) \leq e(\underline{b})$ |
| `endBeforeStart`$(\underline{a}, \underline{b})$ | $x(\underline{a}) \wedge x(\underline{b}) \implies e(\underline{a}) \leq s(\underline{b})$ |
| `endBeforeEnd`$(\underline{a}, \underline{b})$ | $x(\underline{a}) \wedge x(\underline{b}) \implies e(\underline{a}) \leq e(\underline{b})$ |
| `startAtStart`$(\underline{a}, \underline{b})$ | $x(\underline{a}) \wedge x(\underline{b}) \implies s(\underline{a}) = s(\underline{b})$ |
| `startAtEnd`$(\underline{a}, \underline{b})$ | $x(\underline{a}) \wedge x(\underline{b}) \implies s(\underline{a}) = e(\underline{b})$ |
| `endAtStart`$(\underline{a}, \underline{b})$ | $x(\underline{a}) \wedge x(\underline{b}) \implies e(\underline{a}) = s(\underline{b})$ |
| `endAtEnd`$(\underline{a}, \underline{b})$ | $x(\underline{a}) \wedge x(\underline{b}) \implies e(\underline{a}) = e(\underline{b})$ |

**Table 5.1.** Semantics of the different temporal constraints.

All temporal constraints are aggregated into a network in order to have a better propagation [DMP91]. Instead of having a bunch of independent constraints, they are considered as a global constraint.

**Span Constraint** `span`$(a_0, \{a_1, \ldots, a_n\})$ [LR08] states that, if activity $a_0$ is executed, it spans over all executed activities from the set $\{a_1, \ldots, a_n\}$. In other words, $a_0$ starts together with the first executed activity from the set and ends together with the last one. The activity $a_0$ is not executed if and

only if no activity of the set is executed. More formally, let $\underline{a_0}, \underline{a_1}, \ldots, \underline{a_n}$ be a set of fixed activities. Then, $\texttt{span}(\underline{a_0}, \{\underline{a_1}, \ldots, \underline{a_n}\})$ holds if and only if:

$$\neg x(\underline{a_0}) \equiv \forall i \in [1,n],\ \neg x(\underline{a_i}) \tag{5.1}$$

$$x(\underline{a_0}) \equiv \begin{cases} \exists i \in [1,n],\ x(\underline{a_i}) \\ s(\underline{a_0}) = \min_{i \in [1,n], x(\underline{a_i})} s(\underline{a_i}) \\ e(\underline{a_0}) = \max_{i \in [1,n], x(\underline{a_i})} e(\underline{a_i}) \end{cases} \tag{5.2}$$

**Alternative Constraint** $\texttt{alternative}(a_0, \{a_1, \ldots, a_n\})$ [LR08] models an exclusive alternative between the activities of $\{a_1, \ldots, a_n\}$. If activity $a_0$ is executed, then exactly one of activities from the set $\{a_1, \ldots, a_n\}$ is executed and $a_0$ starts and ends together with the chosen activity. Activity $a_0$ is not executed if and only if none of activities in $\{a_1, \ldots, a_n\}$ is executed. More formally, $\texttt{alternative}(\underline{a_0}, \{\underline{a_1}, \ldots, \underline{a_n}\})$ holds if and only if:

$$\neg x(\underline{a_0}) \equiv \forall i \in [1,n],\ \neg x(\underline{a_i}) \tag{5.3}$$

$$x(\underline{a_0}) \equiv \exists k \in [1,n] \begin{cases} x(\underline{a_k}) \\ s(\underline{a_0}) = s(\underline{a_k}) \\ e(\underline{a_0}) = e(\underline{a_k}) \\ \forall j \neq k, \neg x(\underline{a_j}) \end{cases} \tag{5.4}$$

**NoOverlap Constraint** `noOverlap(p)` [LRSV09], where $p$ is a sequence of activities, states that any activity in the sequence is constrained to end before the start of the next activity. This constraint is typically useful for modelling disjunctive resources. More formally, the condition for a permutation value $\pi$ of $p$ to satisfy the constraint is defined as:

$$\begin{aligned} noOverlap(\pi) \equiv \forall \underline{a}, \underline{b} \in \underline{A}, \neg x(\underline{a}) \vee \neg x(\underline{b}) \\ \vee \Big( (0 < \pi(\underline{a}) < \pi(\underline{b})) \equiv (e(\underline{a}) \leq s(\underline{b})) \Big) \end{aligned} \tag{5.5}$$

## 5.4 Constraint Programming Model

Let us now come back to the traffic rescheduling problem. Basically, the goal is to schedule adequately trains in order to bring them to their desti-

nation. The decision is then to choose, for each train, which route must be commanded and at what time. In order to illustrate how the problem is modelled, let us introduce the fictive station presented in Figure 5.2.

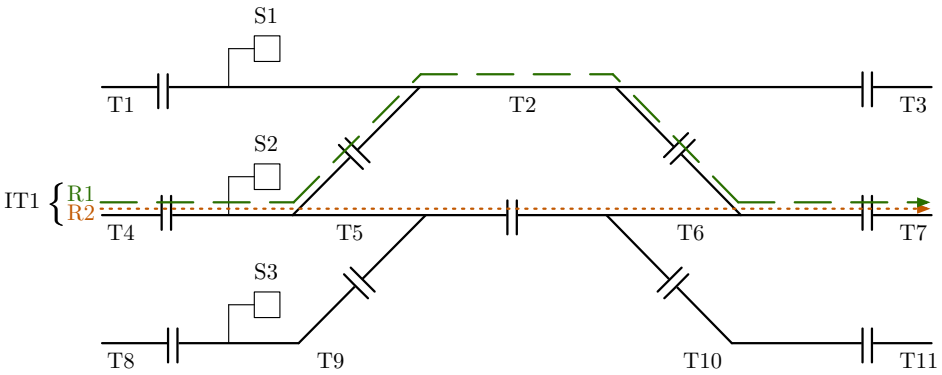



**Figure 5.2.** Layout of a fictive station with two routes and one itinerary.

Let us remember that the *routes* correspond to the paths that trains can follow inside a station in order to reach a destination. We will express them in terms of track segments and signals. For instance, $R1$ is a route going from $T4$ to $T7$ by following the path $[T4, S2, T5, T2, T6]$. The first track segment of a route is always in front of a signal which is initially at a stop aspect and which turns green when the TMS has allowed the train to proceed. The track segment used for the destination is not a part of the route. At this step, all the track segments forming the route after the start signal are reserved and cannot be used by another train. Once a route has been proved, the train can move through it in order to reach its destination. For instance, once a train following route $R1$ has reached $T6$ and is not on the previous track segments anymore, $T5$ and $T2$ can be released in order to allow other trains to use them. Let us notice that this description is a simplification of the complete operations described in Chapter 2, that we do in order to ease the understanding of the CP model.

Furthermore, for the sake of modelling, a new logical component is added. The *itineraries* are the non physical paths from a departure point to an arrival

point. An itinerary can be constituted of one or several routes which can be alternative. For instance, as depicted in Figure 5.2, two routes ($R1$ and $R2$) are possible for accomplishing itinerary $IT1$ from $T4$ to $T7$. In normal situations, a route is often preferred than others, but in case of perturbations, the route causing the less conflicts is preferred.

According to an established timetable, the TMS commands routes in order to ensure the planned traffic. However, in case of real time perturbations, signalling must be handled manually and the actions to perform are decided by human operators. The decision is then to choose, for each train, which route must be commanded and at what time. Let us solve this problem with CP using time-interval variables. The model contains three kinds of activities that are linked together:

- The *itinerary activities*, which define the time interval when a train follows a particular itinerary. Each train has one and only one itinerary activity. We define $A^{t,it}$ as the activity for itinerary $it$ of train $t$.
- The *route activities*, which define the time interval when a train follows a particular route of an itinerary. We define $A^{t,it,r}$ as the activity for route $r$ of itinerary $it$ related to train $t$.
- The *train activities*, which correspond to the movements of a train through the station in order to complete a route. Such activities use the track segments and signals as resources. We define $A_i^{t,it,r}$ as the $i^{th}$ train activity of route $r$ of itinerary $it$ and related to train $t$. Inside a same route, there are as many train activities as the number of elements on the route path. The element can be a track segment or a signal. For instance, by following the example of Figure 5.2, $A_1^{t,IT1,R1}$ is a train activity related to track segment $T4$, $A_2^{t,IT1,R1}$ to signal $S2$, $A_3^{t,IT1,R1}$ to $T5$, etc.

As previously indicated, one particularity of the problem is that some activities are optional. For instance, a train that has to accomplish itinerary $IT1$ can follow either $R1$, or $R2$. If $R1$ is chosen, the activity related to $R2$ will not be executed. Optionality is modelled thanks to time-interval variables. Figure 5.3 presents how the different activities are organised for a train $t$ in the fictive station of Figure 5.2.

Activity $A^{t,IT1}$ is mandatory, it models the fact that the train has to reach its destination. Constraint `alternative` ensures that the train will

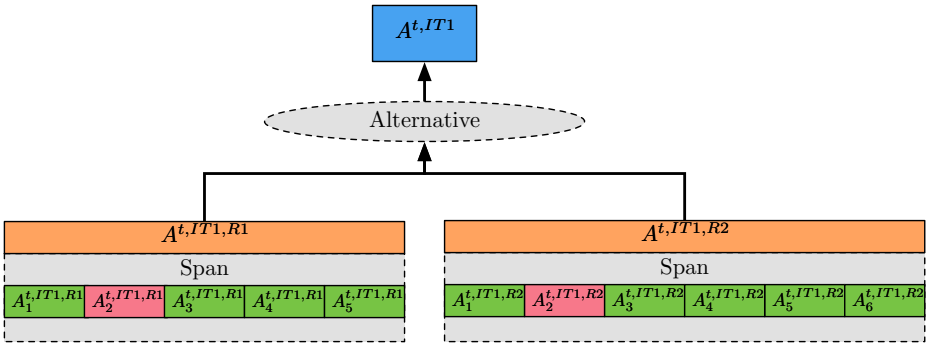



**Figure 5.3.** Breakdown structure of the model using `alternative` and `span` constraints.

follow one and only one route, $R1$ or $R2$. The other route activity is then not executed. Furthermore, the start date and the end date of the chosen route is synchronised with the itinerary activity. Constraint `span` enforces the route activity to be synchronised with the start date of the first track segment activity and with the end date of the last track segment activity. For instance, $A^{t,IT1,R1}$ is synchronised with the start date of $A_1^{t,IT1,R1}$ and the end date of $A_5^{t,IT1,R1}$. Finally, all the aforementioned activities are non-preemptive. Let us now define the different components of the model.

## Parameters

Two entities are involved in the model: the trains and the track segments. Table 5.2 recaps the parameters considered. The *estimated arrival time* of a train is a prediction of its arrival time at the station. The *earliest start time* defines a lower bound on the starting time of a train. In other words, a train cannot start its itinerary before this time. It models the fact that a train cannot leave its platform before the time announced to the passengers. The *planned completion time* is the time announced on the initial schedule. It defines when the train is supposed to arrive at a platform. It is used in the objective function in order to compute the delays generated. The *category* defines the nature of the train. Four categories are considered:

1. Maintenance or special vehicles ($C_1$).
2. Connecting passenger trains ($C_2$).
3. Simple passenger trains ($C_3$).

4. Freight trains ($C_4$).

Such categories are sorted with a decreasing order according to their priority. Special vehicles have then the highest priority and freight trains the lowest.

| Parameter | Name | Meaning |
|---|---|---|
| Speed | $spt_t$ | Speed of $t$ |
| Passengers | $p_t$ | Number of passengers of $t$ |
| Estimated Arrival Time | $eat_t$ | When $t$ arrives to the station |
| Earliest Start Time | $est_t$ | Lower bound on start time of $t$ |
| Planned Completion Time | $pct_t$ | When $t$ must complete its travel |
| Category | $cat_t$ | Category of $t$ |
| Length | $lgt_{ts}$ | Length of $ts$ |

**Table 5.2.** Parameters related to a train $t$ and to a track segment $ts$.

## Decision Variables

As previously said, the problem is to choose, for each train, which route must be commanded and at what time. Such a problem can be seen as a slight variant of a job shop scheduling problem [YN97] where the machines are represented by the track segments and the jobs by train activities. The differences are as follows:

- Some activities are optional. In other words, they may or may not be executed in the final schedule.
- The end of a train activity must be synchronised with the start of the next one. It is also referred as a *no-wait* constraint [MP02].
- A train activity can use more than one resource. Indeed, when a train is on a particular track segment, its current activity uses the current track segment as well as the next ones that are reserved for the route. For instance, let us consider the route activity $A^{t,IT1,R1}$. The related train activities with their resources are $A_1^{t,IT1,R1}$, $A_2^{t,IT1,R1}$, $A_3^{t,IT1,R1}$, $A_4^{t,IT1,R1}$ and $A_5^{t,IT1,R1}$ . The resources used by the activities are $(T4)$, $(T4, S2)$, $(T5, T2, T6)$, $(T2, T6)$ and $(T6)$ respectively. Let us remember that only the track segments located after the start signal are reserved

through the route command. Concerning the initial track segment $T_4$, it is released after the train has passed the start signal.

From a CBS approach, the problem is to reduce each activity domain to a singleton. The decision variables and their domain are, for all trains $t$, itineraries $it$, routes $r$ and indexes $i$:

$$s(A_i^{t,it,r}) \begin{cases} \in [eat_t, horizon] & \text{if } t \text{ is on a track segment.} \\ \in [est_t, horizon] & \text{if } t \text{ is in front of a signal.} \end{cases} \tag{5.6}$$

$$d(A_i^{t,it,r}) \begin{cases} = lgt_{ts}/spd_t & \text{if } t \text{ is on track segment } ts. \\ \in [0, horizon] & \text{if } t \text{ is in front of a signal.} \end{cases} \tag{5.7}$$

$$e(A_i^{t,it,r}) = s(A_i^{t,it,r}) + d(A_i^{t,it,r}) \tag{5.8}$$

$$x(A_i^{t,it,r}) \in \{0, 1\} \tag{5.9}$$

Where $s(A)$, $d(A)$, $e(A)$ and $x(A)$ are the start date, the duration, the end date and the execution status of an activity $A$ respectively. The domain is designed to be as restricted as possible without removing a solution. Equation (5.6) indicates that an activity cannot begin before the *estimated arrival time* of $t$. The upper bound of the start date is defined by the time horizon considered. More details about the horizon chosen is provided in Section 5.5. Equation (5.7) models the time required to achieve the activity. If $t$ is on a track segment, the duration is simply the speed of $t$ divided by the length of the current track segment but if $t$ is in front of a signal, the time that it will have to wait is unknown. Equation (5.8) is an implicit constraint of consistency. Finally, Equation (5.9) states that the activity is optional. Concerning route and itinerary activities, they are linked to train activities through constraints.

## Constraints

Most of the constraints are expressed in terms of a train, an itinerary and a route. Let us express $T$ as the set of trains, $IT_t$ as the set of possible itineraries for $t \in T$, $R_{it}$ the set of possible routes for $it \in IT_t$ and $N_r$ as the number of train activities of a route $r \in R_{it}$. Furthermore, let us state $TS$ as the set of all the track segments in the station.

**Precedence** This constraint (Equation (5.10)) ensures that train activities must be executed in a particular order. It links the end of a train activity $A_i^{t,it,r}$ with the start of $A_{i+1}^{t,it,r}$.

$$\left.\begin{array}{l}\forall t \in T \\ \forall it \in IT_t \\ \forall r \in R_{it} \\ \forall i \in [1, N_r)\end{array}\right\} \quad \texttt{endAtStart}\Big(A_i^{t,it,r}, A_{i+1}^{t,it,r}\Big) \tag{5.10}$$

**Execution Consistency** As previously said, some activities are alternative. If a route is not chosen for a train, none of activities $A_i^{t,it,r}$ will be executed. Otherwise, all of them must be executed. Equation (5.11) states that all the train activities related to the same environment must have the same execution status.

$$\left.\begin{array}{l}\forall t \in T \\ \forall it \in IT_t \\ \forall r \in R_{it} \\ \forall i \in (1, N_r]\end{array}\right\} \quad x(A_1^{t,it,r}) \equiv x(A_i^{t,it,r}) \tag{5.11}$$

**Alternative** This constraint (Equation (5.12)) models an exclusive alternative between a bunch of activities.

$$\left.\begin{array}{l}\forall t \in T \\ \forall it \in IT_t\end{array}\right\} \quad \texttt{alternative}\Big(A^{t,it}, \Big\{A^{t,it,r}\Big| r \in R_{it}\Big\}\Big) \tag{5.12}$$

It means that when $A^{t,it}$ is executed, then exactly one of the route activities must be executed. Furthermore, the start date and the end date of $A^{t,it}$ must be synchronised with the start and end date of the executed route activity. if $A^{t,it}$ is not executed, none of the other activities will be executed. In our case, $A^{t,it}$ is a mandatory activity. It models the fact that each train must reach its destination through an itinerary but for that, it must follow exactly one route.

**Span** This constraint (Equation (5.13)) states that an executed activity must span over a bunch of other executed activities by synchronising its start

date with the earliest start date of other executed activities and its end date with the latest end date.

$$\left.\begin{array}{l}\forall t \in T \\ \forall it \in IT_t \\ \forall r \in R_{it}\end{array}\right\} \quad \texttt{span}\Big(A^{t,it,r}, \Big\{A_i^{t,it,r}\Big| i \in [1, N_r]\Big\}\Big) \tag{5.13}$$

It models the fact that the time taken by a train to complete a route is equal to the time required for crossing each of its components. If the route is not chosen, then no activity will be executed.

**Unary Resource** An important constraint (Equation (5.14)) is that trains cannot move or reserve a track segment that is already used for another train. Each track segment can then be reserved only once at a time.

$$\forall ts \in TS, \ \texttt{noOverlap}\Big(\Big\{A\Big|A \in ACT_{ts}\Big\}\Big) \tag{5.14}$$

$ACT_{ts}$ is the set of all the train activities using track segment $ts$.

**Train Order Consistency** This last constraint (Equation (5.15)) ensures that trains cannot overtake other trains if they are on the same track segment. An illustration of this scenario is presented in Figure 5.4. Even if train $t_2$ has a higher priority than $t_1$, it cannot begin its activities before $t_1$ because $t_1$ has an earlier estimated arrival time. In other words, on each track segment in front of a signal, the start date of the first activity of each train is sorted by the estimated arrival time of the trains.

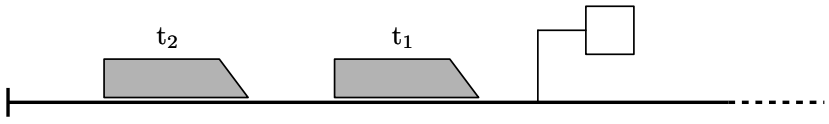



**Figure 5.4.** Two trains waiting on the same track segment.

Let us state $TSB \subset TS$ as the set of all first track segments, $N_{tsb}$ as the number of trains beginning their itinerary on $tsb$, and $(AB_i^{tsb})$ as the

sequence of the first activities of trains $t_i$ beginning on $tsb$ with $i \in [1, N_{tsb}]$. The sequence is ordered by the estimated arrival time of trains $t_i$.

$$\left.\begin{array}{l}\forall tsb \in TSB \\ \forall i \in (1, N_{tsb}]\end{array}\right\} \ \texttt{startBeforeStart}\Big(AB_{i-1}^{tsb}, AB_{i}^{tsb}\Big) \tag{5.15}$$

## Objective Function

The criterion frequently used for the objective function is the cumulated sum of train delays [Rod07]. Let us state $jct_t$ as the *journey completion time* of a train $t$. It corresponds to the end date of the last train activity of $t$. The delay $d_t$ of $t$ is expressed in Equation 5.16.

$$d_t = \max\Big(0, jct_t - pct_t\Big) \tag{5.16}$$

This expression is used to nullify the situation where $t$ is in advance on its schedule. Equation 5.17 presents a first objective function.

$$\min\Big(\sum_{t \in T} d_t\Big) \tag{5.17}$$

However, in real circumstances, railway operators must also consider other parameters such as the number of passengers and the priority of trains. Intuitively, if a passenger train has more passengers than another one, the cost of the delay will be more important. The objective function is then threefold:

1. Scheduling trains according to their priority. For instance, a maintenance vehicle must be scheduled before a passenger train, if possible.
2. Minimising the sum of delays.
3. Maximising the overall passenger satisfaction. The passenger satisfaction decreases if its train is late.

A second objective function can then be expressed (Equation (5.18)).

$$\text{lexMin}\Big(\sum_{t \in C_1} d_t, \sum_{t \in C_2} p_t \times d_t, \sum_{t \in C_3} p_t \times d_t, \sum_{t \in C_4} d_t\Big) \tag{5.18}$$

Where $p_t$ corresponds to the number of passengers of train $t$, as defined in Table 5.2. This equation gives a lexicographical order of trains according to their category from $C1$ to $C4$. For passenger categories ($C_2$ and $C_3$) the delay is expressed by passengers. In this way, the more is the number of passenger, the greater will be the penalty for delays. The objective is then to minimise this expression with regard to its lexicographical order.

### Search Phase

The exploration of the state space is performed with the algorithm of Vilim et al. [VLS15] which combines a Failure-Directed Search with Large Neighborhood Search. The implementation proposed on CP Optimizer V12.6.3 [Lab09] is particularly fitted to deal with conditional time-interval variables, precedence constraints, and optional resources. The search is performed on execution and on start date variables. Concerning the variable ordering, trains are sorted according to their category. For instance, activities related to passenger trains will be assigned before activities of freight trains. The value ordering is let by default.

## 5.5 Experimental Results

This section evaluates the performance of the CP model through different experiments. Concretely, we compare our solution with the solutions obtained with classical dispatching methods on the three case studies. Three systematic strategies are commonly used in practice:

1. *First Come First Served (FCFS)*, which gives priority to the train that has the earliest estimated arrival time.
2. *Highest Delay First Served (HDFS)*, which gives priority to the train that has the earliest planned completion time.
3. *Highest Priority First Served (HPFS)*, which gives priority to a train belonging to the category with the highest priority. The planned completion time is then used as a tie breaker for trains of a same category.

Furthermore, three meta-parameters are considered for the experiments, the *time horizon*, the *decision time* and *the number of trains*. The time

horizon defines an upper bound on the estimated arrival time of trains. According to D'Ariano et al. [DPP07], the practical time horizon for railway managers is usually less than one hour. In our experiments, we considered three time horizons (30 minutes, one hour and two hours). The decision time is the time that railway operators have at disposal for taking a decision. It is highly dependant to the criticality of the situation. However, according to Rodriguez [Rod07], the system must be able to provide an acceptable solution within 3 minutes for practical uses. In our case, it correspond to the time that has the solver for giving a solution. Concerning the number of trains, we considered scenarios having 5, 10, 15, 20, 25 and 30 trains. Homogeneous and heterogeneous traffics are both considered. Finally, other aspects such as the scalability of the approach or the management of critical cases are also analysed.

The implementation of the model has been performed with IBM ILOG CP Optimizer V12.6.3 [Lab09] which is particularly fitted for designing scheduling models [Vil09, SEB+13, HA11]. All the experiments have been realised on a MacBook Pro with a 2.6 GHz Intel Core i5 processor and with a RAM of 16 Go 1600 MHz DDR3 using a 64-Bit HotSpot(TM) JVM 1.8. The optimisation is performed using four workers.

### Homogeneous Traffic

In this first situation, the number of passengers and the category are not considered. Each train has then the same priority and Equation (5.17) is the objective function used. Table 5.3 recaps the experiments performed for Courtrai (Appendix B.3), which is our largest case study.

Each scenario is repeated one hundred times with a random schedule. The different values of the schedule are generated randomly with uniform distributions. For instance, let us consider a schedule from 1pm to 3pm with 10 trains. For each train, we randomly choose its itinerary among the set of possible itineraries, its departure time and its expected arrival time in the interval [1pm, 3pm]. The delay indicated for each strategy corresponds to the arithmetic mean among all the tests. For each scenario, the average, the minimum and the maximum improvement ratio of the CP solution in comparison to the best solution obtained with classical methods is also

| # trains | Average Delay (min.) | | | Improvement Ratio (%) | | | POS | OPT |
|---|---|---|---|---|---|---|---|---|
| | FCFS | HDFS | CP | Mean | Min | Max | $(x/100)$ | $(x/100)$ |
| Horizon of two hours | | | | | | | | |
| 5 | 192.06 | 191.10 | 148.91 | 22.08 | -7.69 | 100.00 | 99 | 97 |
| 10 | 800.40 | 797.82 | 575.04 | 27.92 | 5.13 | 66.08 | 100 | 85 |
| 15 | 1917.95 | 1896.64 | 1341.53 | 29.27 | 1.16 | 58.549 | 100 | 28 |
| 20 | 3457.45 | 3397.03 | 2414.45 | 28.92 | 10.26 | 53.29 | 100 | 0 |
| 25 | 5581.10 | 5632.69 | 3993.04 | 28.45 | 8.58 | 48.37 | 100 | 0 |
| 30 | 8004.84 | 8018.76 | 5714.69 | 28.61 | 14.35 | 43.61 | 100 | 0 |
| Horizon of one hour | | | | | | | | |
| 5 | 225.36 | 228.75 | 172.06 | 23.65 | 0.71 | 100.00 | 100 | 96 |
| 10 | 911.01 | 906.32 | 664.18 | 26.72 | 3.96 | 62.41 | 100 | 83 |
| 15 | 2128.34 | 2104.95 | 1494.67 | 28.99 | 5.93 | 51.29 | 100 | 17 |
| 20 | 3675.72 | 3680.00 | 2612.6 | 28.92 | 8.25 | 55.86 | 100 | 1 |
| 25 | 5971.54 | 6004.81 | 4246.55 | 28.89 | 9.41 | 53.67 | 100 | 0 |
| 30 | 8595.56 | 8576.92 | 6085.51 | 29.05 | 7.89 | 45.51 | 100 | 0 |
| Horizon of 30 minutes | | | | | | | | |
| 5 | 231.16 | 229.06 | 173.95 | 24.06 | 1.49 | 100.00 | 100 | 94 |
| 10 | 950.30 | 929.71 | 692.18 | 25.55 | 3.12 | 64.32 | 100 | 83 |
| 15 | 2145.36 | 2161.20 | 1535.86 | 28.41 | 7.927 | 51.61 | 100 | 14 |
| 20 | 3858.67 | 3888.29 | 2728.0 | 29.30 | 6.63 | 52.50 | 100 | 0 |
| 25 | 6137.02 | 6135.59 | 4320.14 | 29.59 | 7.75 | 53.38 | 100 | 0 |
| 30 | 8863.49 | 8775.84 | 6357.08 | 27.56 | 8.72 | 49.08 | 100 | 0 |

**Table 5.3.** Comparison between CP and classical scheduling approaches for a homogeneous traffic with a decision time of 3 minutes on Courtrai.

indicated. POS indicates the number of tests where the CP approach has improved the solution while OPT indicates the number of tests where CP has reached the optimal solution (notified by the solver). As we can see, CP improves the solution for almost all the tests. The average improvement ratio is above 20% in all the scenarios. Optimum is often reached (more than 75 % of the instances) when 10 trains or less are considered. Experiments on the other case studies are proposed in Appendix E.1. Similar results are observed.

### Heterogeneous Traffic

In this second situation, Equation (5.18) is used for the objective function. The number of passengers and the category are then considered. As for the heterogeneous case, such values are chosen randomly with a uniform distribution. Among the classical approaches, only HPFS deals with an heterogeneous traffic. Table 5.4 recaps the experiments performed for Courtrai. Optimisations are performed sequentially for each category. The time is allocated according to the priority of categories. The allocation is then not

done *a priori* but dynamically according to the time taken by the successive optimisations. In other words, the optimisation is firstly done only for trains of the highest category, and if the decision time is not reached, the optimisation is done for trains of the second category. This process continues until the decision time is reached or until all the categories have been considered.

| # trains | POS ($x$/100) | OPT ($x$/100) | POS ($x$/100) | OPT ($x$/100) | POS ($x$/100) | OPT ($x$/100) |
|---|---|---|---|---|---|---|
| | Two hours | | One hour | | 30 minutes | |
| 5 | 100 | 99 | 100 | 99 | 100 | 99 |
| 10 | 98 | 87 | 99 | 89 | 99 | 87 |
| 15 | 100 | 64 | 100 | 75 | 100 | 75 |
| 20 | 100 | 51 | 100 | 51 | 100 | 52 |
| 25 | 100 | 13 | 100 | 22 | 100 | 26 |
| 30 | 100 | 5 | 100 | 8 | 100 | 7 |

**Table 5.4.** Comparison between CP and HPFS approach for a heterogeneous traffic with a decision time of 3 minutes for Courtrai.

Unlike the previous experiments, we do not compute the improvement ratio, but the number of experiences where CP has improved the solution obtained with HPFS. This choice was motivated by the subjective aspect of defining an improvement ratio for a heterogeneous traffic. For instance, there is no clear preference between decreasing the delay of one train of category $C_1$ or 10 trains with lower priorities. This kind of questions usually requires the consideration of signalling operators. We consider that CP has improved the solution when the sum of delay per category is lexicographically lower than the result provided by HPFS. For a time horizon of one hour and for 100 tests per scenario, CP improves the solution for all the tests, even when the optimum is not reached. Experiments on the other case studies are proposed in Appendix E.2. Similar results are also observed.

### Scalability

This experiment deals with the scalability of the CP model in function of the decision time. We observed that setting the decision time to 10 minutes instead of 3 minutes does not significantly increase the performances. The gain of the improvement ratio is less than 1% for 60 random instances on a homogeneous traffic of 5,10,15,20,25 or 30 trains on Courtrai (10 instances per configuration). We can then conclude that even if the CP approach

gives a feasible and competitive solution within 3 minutes, the quality of the solution does not increase significantly with time.

### Criticality

In some cases, railway operators do not have an available decision time of 180 seconds, they have to react almost instantly because of the criticality of the situation. For this reason, we analysed how the CP model performs with a tiny decision time (10 seconds). To do so, we recorded the number of experiments where the CP approach has improved the solution in comparison to FCFS and HDFS strategies. For the scenarios depicted in Table 5.3, we observed that CP provides a same or better solution in more than 99% of the cases and can then also be used to deal with critical situations.

### Reproducibility

A shortcoming in the literature about this field of research is the lack of reproducibility. To overcome this lack, we decided to provide information about our instances and the tests performed. To do so, we submitted our problem to CSPLIB [GW99], an open-source library of test problems for constraint solvers. It is indexed as Problem 78. The information provided are enough to build a model, perform experiments and to compare them with ours.

## 5.6 Future Work

Possibilities of future work are multiple. First, the model proposed in this section can be improved in order to reflect more accurately the situation. For instance, speed of trains can vary according its position in the station. It is the case when the train is in front of a signal, where it must brake. The fact that trains can occupy several track segments at the same time can also be considered. Such improvements will give a better representation of the real situation and will then reduce the gap between the solution obtained and its practical use. Defining a suitable and pertinent improvement metric for the heterogeneous case is also an open question.

Secondly, the model is based under the assumption that complete and accurate information is provided. However, it might not be the case. For

instance, we are not always sure at what time a train will exactly arrive to the station. This is especially true when a large horizon of time is considered. Instead, the estimated arrival time can be expressed with a normal probability distribution. It is also possible that new unexpected events occur meanwhile (obstruction of tracks, failure of a component, etc.). This stochastic nature of the problem raises new challenges: how can we build robust schedules (i.e. slight deviations from the schedule would not generate important differences) ? how to deal with new information ? how re-optimisation can be done once the information become more precise ? how can we change the model on the fly when some routes become unusable ?

Answering to such questions is not trivial but is of a great interest for practical uses. One idea for dealing with re-optimisation is to use perturbative methods for recomputing new solutions from the first solution obtained. For instance, it can be done using Constraint Based Local Search (CBLS) [VHM09].

Finally, the area considered for the optimisation is limited to a single station. Considering areas covering several stations and analysing the scalability of the approach will also be a great step forward and is a natural follow up of this work.

## 5.7 Summary

Railway operators must deal with the problem of rescheduling the railway traffic in case of real time perturbations in the network. However, the systematic and greedy strategies (FCFS, HDFS and HPFS) currently used for this purpose often give a suboptimal decision. We addressed this issue by proposing a CP model for rescheduling the railway traffic on real time situations. The modelling is based on the recently introduced time-interval variables. Such a structure allows to design the model elegantly with variables and global constraints especially dedicated for scheduling. Finally, an objective function taking into account the heterogeneity of the traffic is proposed. Experiments have shown that a dispatching better than the classical approaches is obtained in less than three minutes in almost all the situations that can occur in a large station, even when the optimum is not reached.

# Chapter 6
# Conclusion

*"In literature and in life we ultimately pursue, not conclusions, but beginnings."*

–Sam Tanenhaus

Nowadays, many countries must deal with the huge development of railway systems. The number of trains, the number of tracks, the complexity of networks have become so important that classical methods ensuring proper operations do not suit anymore. In this context, the development of new methods, more adapted to the current situation, is crucial. It is the problematic addressed by this thesis. The research goal was to design and develop innovative methods in order to ensure the safety, the availability and the fluidity of the railway traffic. To do so, three main contributions have been proposed.

Firstly, Chapter 3 presented how the safety of an interlocking can be efficiently verified. State of the arts methods dealing with the automatic verification of interlocking systems suffer from the state space explosion problem. We proposed to use our knowledge of the railway field in order to accelerate the verification. The contribution is a problem reduction property which can be used for reducing the state space and then speed up the verification provided that a constraint of monotonicity hold. As we saw, this constraint is satisfied in practical situations. Furthermore, we designed a polynomial algorithm, based on this proof, in order to verify application data expressed in SSI. The validity of this method has been corroborated empirically by the successful detection of several errors that were introduced in the application data.

Secondly, Chapter 4 has introduced how the availability of the traffic can be ensured. Besides the safety, a correct interlocking must also prevent trains from being locked in the station because of improper interlocking operations. It will directly ensure the progression of each train in the network. To do so,

a complete and automated approach is proposed. The contribution is a model reflecting the behaviour of an interlocking which can be simulated and then verified using Statistical Model Checking. Although the verification is not exhaustive as for the safety, Statistical Model Checking can give a parametrisable level of confidence on the correctness of the system which is sufficient for the verification of availability. As in Chapter 3, this approach has also been validated empirically by the successful detection of several introduced errors. Beyond the verification of availability, this approach can be considered as a general framework for verifying properties related to interlocking systems.

Finally, Chapter 5 has tackled the fluidity aspect which is beyond the interlocking scope. When safety and availability are ensured, the next step is to do the best for providing a traffic having the least delay as possible. Railway operators must deal with the problem of rescheduling the railway traffic in case of real time perturbations in the network. However, the systematic and greedy strategies currently used often give a suboptimal decision. The contribution is a Constraint Programming model for rescheduling the railway traffic on real time situations. The modelling is based on the recently introduced time-interval variables. Experiments have shown that a dynamic rescheduling, better than the classical approaches, is obtained in less than three minutes in almost all the situations that can occur in a large station, even when the optimum is not reached.

The aforementioned contributions led to four scientific papers [CLSL15, CS16, CLS$^+$17, CS17] that have been published and presented in international conferences. To conclude, let us finally notice that all the work done in this thesis is still in its early stage. Ideas and approaches have been proposed and validated, prototypes have been implemented, possibilities of future work have been stated but the ultimate goal pursued is the integration of these methods into a framework which can be plugged with the existing infrastructure and then be used in practice. New questions may raise and new aspects should certainly be considered. Such an integration is then a natural follow up of this work but is resorts more to the engineering science. The other main challenge is analysing the scalability of the approaches and to adapt them to larger areas covering several interlockings or stations.

# Appendix A
# Inograms Project

Railway operators are faced with competition from road, air, waterway and maritime transport. They need to improve in terms of the globalisation of traffic and the interoperability between operators and infrastructures which are known to be more complex than in other modes.

The ERTMS standard, developed by Europe to meet this requirement, is spreading throughout the world, throwing a spotlight on a demand for evolution towards new functionalities and on the risk of an emergence of extra-European competition in the sector. Inograms is an industrial research project which aims to explore new technological avenues to meet the needs of extra-European ERTMS markets, offering the new functionalities and significant cost reductions which are necessary to protect the competitiveness of products developed in Wallonia.

Inograms encompasses research activities in the fields of systems engineering, information processing, and control-command systems. It supports ERTMS in becoming accepted as the only standardised international system guaranteeing interoperability between manufacturers. It then contributes to improving the recognised position of Wallonia as a centre of excellence in ERTMS and assist Wallonia's industry in dealing with competition from outside Europe stimulated by the growth of ERTMS in extra-European markets. Given the large scope of this project, it is divided into seven work package, each of them being dedicated to a particular aspect of railway transportation, such as interlockings (WP1), systems engineering (WP2), energy harvesting (WP3), data processing (WP4), track activity monitoring (WP5), hybridisation of technologies (WP6) and automatic train operation (WP7).

This description is slightly adapted from the official description of the project, issued by *Logistic in Wallonia* Cluster.

# Appendix B
# Case Studies

## B.1 Namêche (Belgium)

Namêche is a station in Belgium located near Namur in Wallonia. It is a small station, composed of 4 tracks, 13 tracks segments, 7 points, 7 signals, 14 routes, 7 immobilisation zones and 26 subroutes. There is no bidirectional locking. It is controlled by a single interlocking.

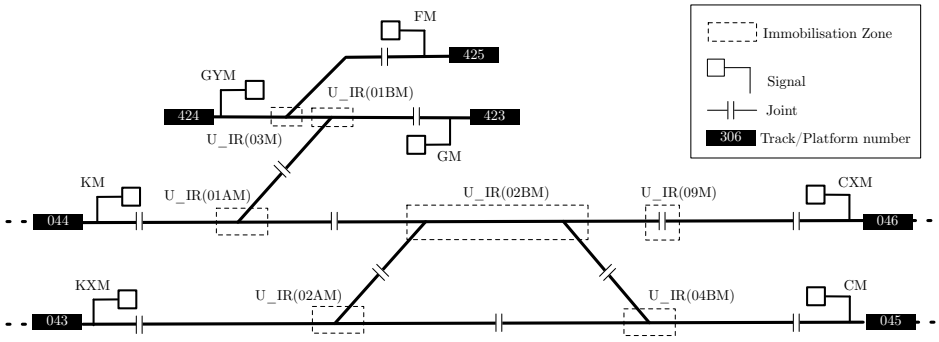



**Figure B.1.** Layout of Namêche.

## B.2 Braine l'Alleud (Belgium)

Braine l'Alleud is a medium sized station located in the center of Belgium in Wallonia. It is located on the direct line between Charleroi and Brussels. It is composed of 4 tracks, 17 track segments, 12 points, 12 signals, 32 routes, 10 immobilisation zones, 48 subroutes and 4 bidirectional locking mechanisms. As Namêche, it is fully controlled by a single interlocking.

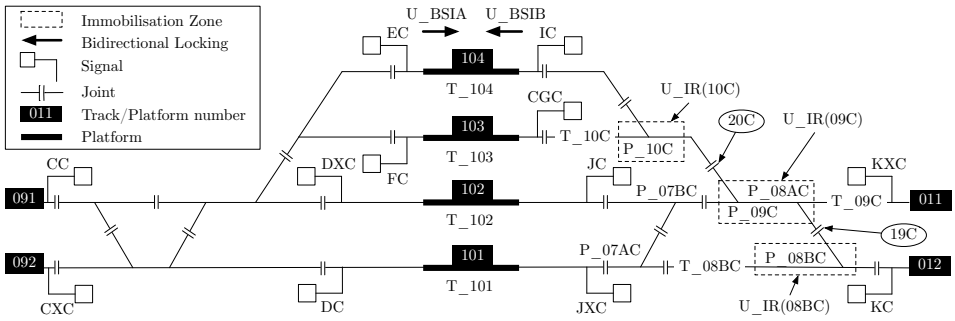



**Figure B.2.** Layout of Braine l'Alleud.

## B.3 Courtrai (Belgium)

Courtrai, or *Kortrijk*, is a large Belgian station located in the Flemish province of West Flanders. The station is split into three zones (LK6, LK7 and LK8) controlled each by a different interlocking. The three interlockings communicate together in order to share information about the station state. Only LK7 is considered here. This area is composed of 6 tracks, 19 track segments, 26 points, 24 signals (14 of them are fictive), 70 routes, 10 immobilisation zones and 72 subroutes. There are 4 bidirectional locking mechanisms shared with other zones but none only defined in LK7. Unlike Namêche and Braine l'Alleud, Courtrai has full paths. There are 96 full paths, which are composed of at most 2 different routes.

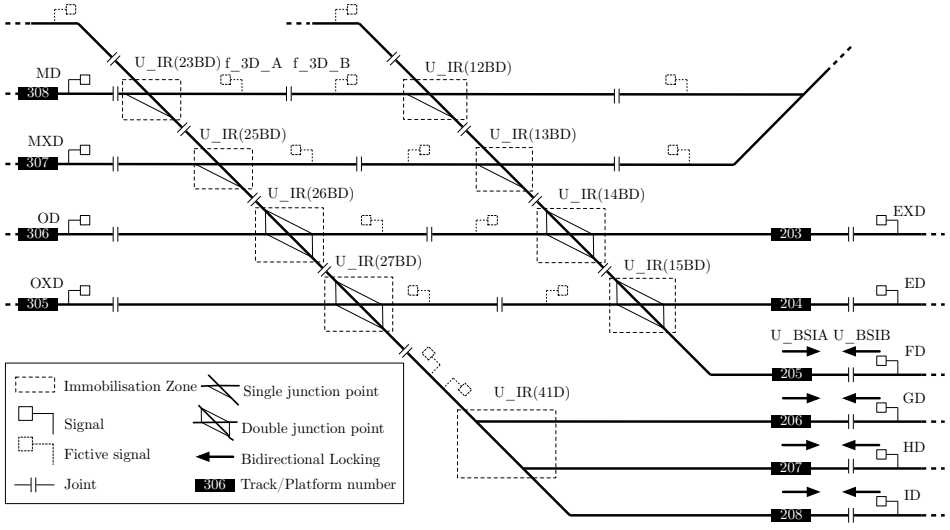



**Figure B.3.** Layout of Courtrai.

# Appendix C
# Application Data Grammars

## C.1 Structure Definition

**Grammar C.1** (Variables, values and expressions in SSI)**.**

| | | |
|---|---|---|
| ⟨*route*⟩ | ::= | `R_`⟨*id*⟩`_`⟨*id*⟩ |
| ⟨*subroute*⟩ | ::= | `U_`⟨*id*⟩`_`⟨*id*⟩ |
| ⟨*track_segment*⟩ | ::= | `T_`⟨*id*⟩ |
| ⟨*immobilisation_zone*⟩ | ::= | `U_IR`$\Big($⟨*id*⟩$\Big)$ |
| ⟨*point*⟩ | ::= | `P_`⟨*id*⟩ |
| ⟨*signal*⟩ | ::= | `S_`⟨*id*⟩ |
| ⟨*bidirectional_locking*⟩ | ::= | `U_BSI(A\|B)`$\Big($⟨*id*⟩$\Big)$ |
| ⟨*variable*⟩ | ::= | ⟨*route*⟩ \| ⟨*subroute*⟩ \| ⟨*immobilisation_zone*⟩ \| ⟨*track_segment*⟩ \| ⟨*bidirectional_locking*⟩ \| ⟨*point*⟩ \| ⟨*signal*⟩ |
| ⟨*value*⟩ | ::= | `f` \| `l` \| `s` \| `xs` \| `c` \| `o` \| `cfn` \| `cfr` \| `cn` \| `cr` \| `cdn` \| `cdr` \| `stop` \| `proceed` |
| ⟨*condition*⟩ | ::= | ⟨*variable*⟩ ⟨*value*⟩ |
| ⟨*assignment*⟩ | ::= | ⟨*variable*⟩ ⟨*value*⟩ |

Where ⟨*id*⟩ can be any string.

## C.2 PRR File

It defines the conditions that must be satisfied for granting a route command. The actions that the interlocking must perform when the request is accepted are also defined inside.

**Grammar C.2** (PRR File)**.**

⟨*PRR*⟩ ::= {⟨*route_request*⟩}

⟨*route_request*⟩ ::= `Q*`(⟨*route*⟩)
`if` {⟨*condition*⟩}
`then` {⟨*assignment*⟩}
[⟨*UBSI_expression*⟩]

⟨*UBSI_expression*⟩ ::= `if` ⟨*bidirectional_locking*⟩ `f`
`then` ⟨*bidirectional_locking*⟩ `l`

## C.3 PFM File

It includes all the conditions that must be ensured before moving a point to its normal or its reverse position.

**Grammar C.3** (PFM File)**.**

| | | |
|---|---|---|
| $\langle PFM \rangle$ | ::= | $\{\langle point\_request \rangle\}$ |
| $\langle point\_request \rangle$ | ::= | $*\langle point \rangle$`N` $\langle condition \rangle$ |
| | | $*\langle point \rangle$`R` $\langle condition \rangle$ |

## C.4 FOP File

It defines the necessary conditions for releasing components.

**Grammar C.4** (FOP File)**.**

| | | |
|---|---|---|
| ⟨*FOP*⟩ | ::= | {⟨*subroute_release*⟩ \| ⟨*UIR_release*⟩ \| ⟨*UBSI_release*⟩} |
| ⟨*subroute_release*⟩ | ::= | ⟨*subroute*⟩ `f if` ⟨*condition*⟩ |
| ⟨*UIR_release*⟩ | ::= | `if` ⟨*immobilisation_zone*⟩ `l` `then if` ⟨*condition*⟩ `then` ⟨*assignment*⟩ |
| ⟨*UBSI_release*⟩ | ::= | ⟨*bidirectional_locking*⟩ `f if` ⟨*condition*⟩ |

## C.5 OPT File

It contains the outputs of the interlocking, as the description of the life cycle of a route from their command to their releasing. It is used to know when the start signal of a route can be set at a proceed state.

**Grammar C.5** (OPT File)**.**

| | | |
|---|---|---|
| ⟨*OPT*⟩ | ::= | {⟨*route_life_step*⟩} |
| ⟨*route_life_step*⟩ | ::= | `if` ⟨*condition*⟩<br>`then` ⟨*assignment*⟩<br>`else` ⟨*assignment*⟩ |

# Appendix D
# Application Data Errors

## D.1 Wrong Position for a Point

### Namêche

```
1  *Q_R(GYM_423)
2     if     (...)
3     then   R_GYM_423 s,
4            ~~P_01BM cr~~ P_01BM cn,
5            P_03M cr, P_01AM cr, (...)
```

**Listing D.8.** Point moved in a wrong position when commanding Route `R_GYM_423`.

### Braine l'Alleud

```
1  *Q_R(DC_091)
2     if     (...)
3     then   R_DC_091 s,
4            P_01AC cr, P_01BC cr,  P_02BC cr,
5            ~~P_02AC cr~~ P_02AC cn, (...)
```

**Listing D.9.** Point moved in a wrong position when commanding Route `R_DC_091`.

### Courtrai

```
1  *Q_R(7D_3D)
2     if     (...)
3     then   R_7D_3D s,
4            P_12BD cn, P_13BD cn,
5            ~~P_13AD cn~~ P_13AD cr, (...)
```

**Listing D.10.** Point moved in a wrong position when commanding Route `R_7D_3D`.

## D.2 Subroute not Properly Locked

### Namêche

```
1  *Q_R(KM_045)
2     if     (...)
3     then   R_KM_045 s
4            P_01AM cr, P_02BM cr, P_04AM cr,
5            P_04BM cr, P_01BM cr, P_02AM cr
6            U_IR(01AM) l, U_IR(02BM) l, U_IR(04BM) l,
7            U_KM_07M l, U_07M_04M l, ~~U_04M_CM l~~
```

**Listing D.11.** Subroute not properly locked when commanding Route `R_KM_045`.

### Braine l'Alleud

```
1  *Q_R(CXC_102)
2     if     (...)
3     then   R_CXC_102 s,
4            P_03C cr,
5            P_01BC cn, P_02AC cn,
6            P_02BC cn, P_01AC cn,
7            U_IR(01BC) l, U_IR(02BC) l,
8            ~~U_CXC_13C l, U_13C_DXC l~~
```

**Listing D.12.** Subroute not properly locked when commanding Route `R_CXC_102`.

### Courtrai

```
1  *Q_R(4D_307)
2     if     (...)
3     then   R_4D_307 s,
4            P_25BD cn,  P_26AD cn,
5            P_25AD cn, P_26BD cn,
6            U_IR(25BD) l,
7            ~~U_4D_MXD l~~
```

**Listing D.13.** Subroute not properly locked when commanding Route `R_4D_307`.

## D.3 Missing Conditions for Releasing a Subroute

### Namêche

```
1  U_02M_KXM f if
2     U_09M_02M f,
3     T_02AM c
```

**Listing D.14.** Missing conditions for releasing Subroute `U_02M_KXM`.

### Braine l'Alleud

```
1  U_18C_KXC f if
2     U_16C_18C f, U_JC_18C f,
3     T_09C c
```

**Listing D.15.** Missing conditions for releasing Subroute `U_18C_KXC`.

### Courtrai

```
1  U_26D_5D f if
2     U_23D_26D f,
3     U_MD_26D f,
4     U_MXD_26D f,
5     T_26BD c
```

**Listing D.16.** Missing conditions for releasing Subroute `U_26D_5D`.

## D.4 Missing Conditions on a Route Command

### Namêche

```
1  *Q_R(GM_044)
2     if     R_GM_044 xs
3            P_01AM cfn, P_01BM cfn, P_03M cfn,
4            U_IR(01AM) f, U_IR(01BM) f
5     then (...)
```

**Listing D.17.** Missing conditions for commanding Route `R_GM_044`.

### Braine l'Alleud

```
1  *Q_R(JXC_011)
2     if     R_JXC_011 xs
3            P_07AC cfn, P_07BC cfn, P_09C cfn,
4            P_08AC cfn, P_08BC cfn,
5            U_IR(07AC) f, U_IR(07BC) f, U_IR(09C) f
6     then   (...)
```

**Listing D.18.** Missing conditions for commanding Route `R_JXC_011`.

### Courtrai

```
1  *Q_R(12D_203)
2     if     R_12D_203 xs
3            P_12BD cfr, P_13AD cfr, P_13BD cfr,
4            P_14AD cfr, P_14BD cfr,
5            P_15AD cfn, P_15BD cfn,
6            U_IR(12BD) f, U_IR(13BD) f, U_IR(14BD) f
7     then   (...)
```

**Listing D.19.** Missing conditions for commanding Route `R_12D_203`.

## D.5 Additional Conditions for Releasing a Component

### Namêche

```
1  U_09M_07M f if
2     U_CXM_09M f,
3     U_07M_KM f,
4     T_02BM c,
5     T_01AM o
```

**Listing D.20.** Additional conditions for releasing Subroute `U_09M_07M`.

### Braine l'Alleud

```
1  U_JXC_17C f if
2     R_JXC_012 xs,
3     U_IR(07AC) f,
4     T_07AC c
```

**Listing D.21.** Additional conditions for releasing Subroute `U_JXC_17C`.

### Courtrai

```
1  U_ED_14D f if
2     R_ED_12D xs, R_ED_3D xs, R_ED_4D xs,
3     U_14D_EXD f,
4     T_15BD c, T_14BD c
```

**Listing D.22.** Additional conditions for releasing Subroute `U_ED_14D`.

## D.6 Bidirectional Locking not Properly Locked

### Namêche

There is no bidirectional locking in Namêche.

### Braine l'Alleud

```
1  *Q_R(KC_102)
2     if     (...)
3     then   R_KC_102 s
4            P_07BC cr, P_08AC cr, P_08BC cr, P_07AC cr,
5            P_09C cn,
6            U_IR(07BC) l, U_IR(08BC) l, U_IR(09C) l,
7            U_KC_19C l, U_19C_18C l, U_18C_JC l,
8            ~~if U_BSIA(102) f then U_BSIB(102) l~~
```

**Listing D.23.** Bidirectional locking not properly locked when commanding Route R_KC_102.

### Courtrai

There is no bidirectional locking in Section LK7 of Courtrai.

## D.7 Inconsistency in Route Proving

### Namêche

```
1  *R_KM_045
2     P_01AM cdr, P_02BM cdr, P_04AM cdr,
3     ~~P_04BM cdr~~ P_04BM cdn,
4     U_IR(01AM) l, U_IR(02BM) l, U_IR(04BM) l
5     T_01AM c, T_02BM c, T_04BM c
6     (...)
```

**Listing D.24.** Inconsistency in the proving of Route `R_KM_045`.

### Braine l'Alleud

```
1  *R_CXC_101
2     P_01BC cdn,
3     ~~P_02AC cdr~~ P_02AC cdn,
4     U_IR(01BC) l,
5     U_BSIB(101) f,
6     T_01BC c, T_101 c,
7     (...)
```

**Listing D.25.** Inconsistency in the proving of Route `R_CXC_101`.

### Courtrai

```
1  *R_MD_3D
2     P_23BD cdn, ~~P_25AD cdn~~ P_25AD cdr,
3     U_IR(23BD) l,
4     T_23BD c
5     (...)
```

**Listing D.26.** Inconsistency in the proving of Route `R_MD_3D`.

# Appendix E
# Benchmarks

## E.1 Homogeneous Traffic

### E.1.1 Namêche

| # trains | Average Delay (min.) | | | Improvement Ratio (%) | | | POS ($x$/100) | OPT ($x$/100) |
|---|---|---|---|---|---|---|---|---|
| | FCFS | HDFS | CP | Mean | Min | Max | | |
| Horizon of two hours | | | | | | | | |
| 5 | 38.88 | 40.27 | 31.22 | 19.70 | 0.00 | 100.00 | 100 | 100 |
| 10 | 236.49 | 238.54 | 182.49 | 22.83 | 3.29 | 55.92 | 100 | 100 |
| 15 | 649.56 | 642.07 | 486.86 | 24.18 | 5.98 | 56.19 | 100 | 100 |
| 20 | 1206.16 | 1190.51 | 925.18 | 22.29 | 7.19 | 44.05 | 100 | 91 |
| 25 | 2155.13 | 2122.59 | 1656.8 | 21.94 | 10.07 | 37.28 | 100 | 18 |
| 30 | 3075.02 | 3039.92 | 2400.56 | 21.03 | 10.06 | 35.89 | 100 | 0 |
| Horizon of one hour | | | | | | | | |
| 5 | 59.07 | 61.47 | 47.32 | 19.89 | 0.00 | 100.00 | 100 | 100 |
| 10 | 313.43 | 316.37 | 247.05 | 21.18 | 3.78 | 73.21 | 100 | 100 |
| 15 | 755.45 | 758.97 | 596.44 | 21.05 | 4.65 | 53.10 | 100 | 100 |
| 20 | 1404.53 | 1396.52 | 1122.44 | 19.63 | 7.20 | 28.53 | 100 | 87 |
| 25 | 2425.92 | 2407.15 | 1909.55 | 20.67 | 8.481 | 31.51 | 100 | 9 |
| 30 | 3487.68 | 3414.48 | 2766.43 | 18.98 | 6.84 | 31.37 | 100 | 0 |
| Horizon of 30 minutes | | | | | | | | |
| 5 | 65.44 | 65.85 | 50.74 | 22.46 | 2.17 | 100.00 | 100 | 100 |
| 10 | 334.14 | 335.24 | 264.69 | 20.78 | 2.44 | 44.54 | 100 | 100 |
| 15 | 818.71 | 819.57 | 641.92 | 21.59 | 5.97 | 40.40 | 100 | 99 |
| 20 | 1543.83 | 1554.92 | 1220.98 | 20.91 | 5.47 | 34.67 | 100 | 88 |
| 25 | 2457.18 | 2469.57 | 1957.49 | 20.34 | 8.36 | 31.30 | 100 | 7 |
| 30 | 3649.87 | 3659.01 | 2917.92 | 20.05 | 7.30 | 33.10 | 100 | 0 |

**Table E.1.** Comparison between CP and classical scheduling approaches for a homogeneous traffic with a decision time of 3 minutes on Namêche.

### E.1.2 Braine l'Alleud

| # trains | Average Delay (min.) | | | Improvement Ratio (%) | | | POS | OPT |
|---|---|---|---|---|---|---|---|---|
| | FCFS | HDFS | CP | Mean | Min | Max | $(x/100)$ | $(x/100)$ |
| Horizon of two hours | | | | | | | | |
| 5 | 32.16 | 33.74 | 27.68 | 13.93 | 0.00 | 100.00 | 100 | 100 |
| 10 | 199.20 | 200.80 | 152.02 | 23.68 | 4.55 | 50.64 | 100 | 100 |
| 15 | 555.21 | 550.43 | 412.35 | 25.09 | 6.58 | 43.23 | 100 | 100 |
| 20 | 1091.13 | 1103.64 | 822.69 | 24.60 | 8.76 | 44.29 | 100 | 76 |
| 25 | 2021.35 | 2003.00 | 1511.57 | 24.53 | 9.41 | 40.86 | 100 | 2 |
| 30 | 2945.43 | 2921.32 | 2200.76 | 24.67 | 7.75 | 37.32 | 100 | 0 |
| Horizon of one hour | | | | | | | | |
| 5 | 51.47 | 51.59 | 40.41 | 21.49 | 0.00 | 100.00 | 100 | 100 |
| 10 | 275.09 | 282.21 | 216.44 | 21.32 | 3.12 | 60.00 | 100 | 100 |
| 15 | 711.54 | 718.07 | 542.06 | 23.82 | 3.60 | 46.44 | 100 | 100 |
| 20 | 1325.53 | 1336.11 | 1003.9 | 24.26 | 5.77 | 45.37 | 100 | 30 |
| 25 | 2210.68 | 2243.30 | 1709.52 | 22.67 | 2.36 | 37.70 | 100 | 0 |
| 30 | 3255.52 | 3254.38 | 2509.98 | 22.87 | 0.53 | 36.65 | 100 | 0 |
| Horizon of 30 minutes | | | | | | | | |
| 5 | 56.82 | 57.48 | 43.88 | 22.77 | 0.00 | 100.00 | 100 | 100 |
| 10 | 290.51 | 302.48 | 227.44 | 21.71 | 3.83 | 43.79 | 100 | 100 |
| 15 | 762.12 | 779.67 | 583.13 | 23.49 | -2.91 | 45.10 | 99 | 99 |
| 20 | 1450.54 | 1474.12 | 1107.54 | 23.65 | 5.72 | 47.42 | 100 | 46 |
| 25 | 2337.98 | 2369.70 | 1797.85 | 23.10 | -1.11 | 38.71 | 99 | 0 |
| 30 | 3494.44 | 3540.45 | 2709.84 | 22.45 | 6.63 | 33.78 | 100 | 0 |

**Table E.2.** Comparison between CP and classical scheduling approaches for a homogeneous traffic with a decision time of 3 minutes on Braine l'Alleud.

### E.1.3 Courtrai

| # trains | Average Delay (min.) | | | Improvement Ratio (%) | | | POS | OPT |
|---|---|---|---|---|---|---|---|---|
| | FCFS | HDFS | CP | Mean | Min | Max | ($x$/100) | ($x$/100) |
| Horizon of two hours | | | | | | | | |
| 5 | 192.06 | 191.10 | 148.91 | 22.08 | -7.69 | 100.00 | 99 | 97 |
| 10 | 800.40 | 797.82 | 575.04 | 27.92 | 5.13 | 66.08 | 100 | 85 |
| 15 | 1917.95 | 1896.64 | 1341.53 | 29.27 | 1.16 | 58.549 | 100 | 28 |
| 20 | 3457.45 | 3397.03 | 2414.45 | 28.92 | 10.26 | 53.29 | 100 | 0 |
| 25 | 5581.10 | 5632.69 | 3993.04 | 28.45 | 8.58 | 48.37 | 100 | 0 |
| 30 | 8004.84 | 8018.76 | 5714.69 | 28.61 | 14.35 | 43.61 | 100 | 0 |
| Horizon of one hour | | | | | | | | |
| 5 | 225.36 | 228.75 | 172.06 | 23.65 | 0.71 | 100.00 | 100 | 96 |
| 10 | 911.01 | 906.32 | 664.18 | 26.72 | 3.96 | 62.41 | 100 | 83 |
| 15 | 2128.34 | 2104.95 | 1494.67 | 28.99 | 5.93 | 51.29 | 100 | 17 |
| 20 | 3675.72 | 3680.00 | 2612.6 | 28.92 | 8.25 | 55.86 | 100 | 1 |
| 25 | 5971.54 | 6004.81 | 4246.55 | 28.89 | 9.41 | 53.67 | 100 | 0 |
| 30 | 8595.56 | 8576.92 | 6085.51 | 29.05 | 7.89 | 45.51 | 100 | 0 |
| Horizon of 30 minutes | | | | | | | | |
| 5 | 231.16 | 229.06 | 173.95 | 24.06 | 1.49 | 100.00 | 100 | 94 |
| 10 | 950.30 | 929.71 | 692.18 | 25.55 | 3.12 | 64.32 | 100 | 83 |
| 15 | 2145.36 | 2161.20 | 1535.86 | 28.41 | 7.927 | 51.61 | 100 | 14 |
| 20 | 3858.67 | 3888.29 | 2728.0 | 29.30 | 6.63 | 52.50 | 100 | 0 |
| 25 | 6137.02 | 6135.59 | 4320.14 | 29.59 | 7.75 | 53.38 | 100 | 0 |
| 30 | 8863.49 | 8775.84 | 6357.08 | 27.56 | 8.72 | 49.08 | 100 | 0 |

**Table E.3.** Comparison between CP and classical scheduling approaches for a homogeneous traffic with a decision time of 3 minutes on Courtrai.

## E.2 Heterogeneous Traffic

### E.2.1 Namêche

| # trains | POS ($x$/100) | OPT ($x$/100) | POS ($x$/100) | OPT ($x$/100) | POS ($x$/100) | OPT ($x$/100) |
|---|---|---|---|---|---|---|
| | Two hours | | One hour | | 30 minutes | |
| 5 | 100 | 100 | 100 | 100 | 100 | 100 |
| 10 | 100 | 100 | 99 | 100 | 100 | 100 |
| 15 | 100 | 100 | 100 | 97 | 100 | 98 |
| 20 | 100 | 96 | 100 | 83 | 100 | 92 |
| 25 | 100 | 61 | 100 | 67 | 100 | 65 |
| 30 | 100 | 31 | 100 | 38 | 100 | 21 |

**Table E.4.** Comparison between CP and HPFS approach for a heterogeneous traffic with a decision time of 3 minutes for Namêche.

### E.2.2 Braine l'Alleud

| # trains | POS ($x$/100) | OPT ($x$/100) | POS ($x$/100) | OPT ($x$/100) | POS ($x$/100) | OPT ($x$/100) |
|---|---|---|---|---|---|---|
| | Two hours | | One hour | | 30 minutes | |
| 5 | 100 | 100 | 100 | 100 | 100 | 100 |
| 10 | 100 | 100 | 100 | 100 | 100 | 100 |
| 15 | 99 | 98 | 99 | 99 | 98 | 98 |
| 20 | 100 | 86 | 99 | 81 | 97 | 83 |
| 25 | 99 | 45 | 98 | 48 | 98 | 52 |
| 30 | 99 | 36 | 99 | 28 | 98 | 25 |

**Table E.5.** Comparison between CP and HPFS approach for a heterogeneous traffic with a decision time of 3 minutes for Braine l'Alleud.

### E.2.3 Courtrai

| # trains | POS ($x$/100) | OPT ($x$/100) | POS ($x$/100) | OPT ($x$/100) | POS ($x$/100) | OPT ($x$/100) |
|---|---|---|---|---|---|---|
| | Two hours | | One hour | | 30 minutes | |
| 5 | 100 | 99 | 100 | 99 | 100 | 99 |
| 10 | 98 | 87 | 99 | 89 | 99 | 87 |
| 15 | 100 | 64 | 100 | 75 | 100 | 75 |
| 20 | 100 | 51 | 100 | 51 | 100 | 52 |
| 25 | 100 | 13 | 100 | 22 | 100 | 26 |
| 30 | 100 | 5 | 100 | 8 | 100 | 7 |

**Table E.6.** Comparison between CP and HPFS approach for a heterogeneous traffic with a decision time of 3 minutes for Courtrai.

# Bibliography


[AA08] Mare Antoni and Nadia Ammad, *Formal Validation Method and Tools for French Computerized Railway Interlocking Systems*, Railway Condition Monitoring, 2008 4th IET International Conference on, IET, 2008, pp. 1–10.

[ABF+16] Alexandre Arnold, Massimo Baleani, Alberto Ferrari, Marco Marazza, Valerio Senni, Axel Legay, Jean Quilbeuf, and Christoph Etzien, *An Application of SMC to Continuous Validation of Heterogeneous Systems*, Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016, pp. 76–85.

[ABH+97] Rajeev Alur, Robert K. Brayton, Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani, *Partial-Order Reduction in Symbolic State Space Exploration*, International Conference on Computer Aided Verification, Springer, 1997, pp. 340–351.

[AD94] Rajeev Alur and David L. Dill, *A Theory of Timed Automata*, Theoretical Computer Science **126** (1994), no. 2, 183–235.

[AIM+96] A. M. Amendola, L. Impagliazzo, P. Marmo, G. Mongardi, G. Sartore, and A. Trasporti, *Architecture and Safety Requirements of the ACC Railway Interlocking System*, Proceedings of IEEE International Computer Performance and Dependability Symposium, Sep 1996, pp. 21–29.

[All83] James F. Allen, *Maintaining Knowledge about Temporal Intervals*, Communications of the ACM **26** (1983), no. 11, 832–843.

[Anu09] S. V. Anunchai, *Verification of Railway Interlocking Tables using Coloured Pertri Nets*, Proceedings of the 10th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, 2009.

[AO08] Paul Ammann and Jeff Offutt, *Introduction to Software Testing*, Cambridge University Press, 2008.

[AV13] Robert Abo and Laurent Voisin, *Data Formal Validation of Railway Safety-Related Systems: Implementing the OVADO Tool*, Towards a Formal Methods Body of Knowledge for Railway Control and Safety Systems (2013), 27.

[AWNO85] Katsuji Akita, Toshikatsu Watanabe, Hideo Nakamura, and Ikumasa Okumura, *Computerized Interlocking System for Railway Signaling Control: SMILE*, IEEE Transactions on Industry applications (1985), no. 3, 826–834.

[BBB+10] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Caillaud, Benoît Delahaye, and Axel Legay, *Statistical Abstraction and Model-Checking of Large Heterogeneous Systems*, Formal Techniques for Distributed Systems, Springer, 2010, pp. 32–46.

[BBR07] Jiri Barnat, Lubos Brim, and Petr Rockai, *Scalable Multi-core LTL Model-Checking*, SPIN, vol. 7, Springer, 2007, pp. 187–203.

[BBS86] Lenore Blum, Manuel Blum, and Mike Shub, *A Simple Unpredictable Pseudo-Random Number Generator*, SIAM Journal on Computing **15** (1986), no. 2, 364–383.

[BC07] Roman Barták and Ondrej Cepek, *Temporal Networks with Alternatives: Complexity and Model*, FLAIRS Conference, 2007, pp. 641–646.

[BCL+15] Simon Busard, Quentin Cappart, Christophe Limbrée, Charles Pecheur, and Pierre Schaus, *Verification of railway interlocking systems*, Proceedings 4th International Workshop on Engineering Safety and Security Systems, vol. 184, 2015, p. 19.

[BCLS13] Benoît Boyer, Kevin Corre, Axel Legay, and Sean Sedwards, *PLASMA-Lab: A Flexible, Distributable Statistical*

*Model Checking Library*, Quantitative Evaluation of Systems, Springer, 2013, pp. 160–164.

[BDBK09] David C Black, Jack Donovan, Bill Bunton, and Anna Keist, *SystemC: From the Ground Up*, vol. 71, Springer Science & Business Media, 2009.

[BDCBVL04] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem, *The State of the Art of Nurse Rostering*, Journal of Scheduling **7** (2004), no. 6, 441–499.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine, *Kronos: A Model-Checking Tool for Real-time Systems*, International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, Springer, 1998, pp. 298–302.

[BF99] J. Christopher Beck and Mark S. Fox, *Scheduling Alternative Activities*, AAAI/IAAI, Citeseer, 1999, pp. 680–687.

[BL05] Michael Butler and Michael Leuschel, *Combining CSP and B for Specification and Property Verification*, FM 2005: Formal Methods, Springer, 2005, pp. 221–236.

[BLPN12] Philippe Baptiste, Claude Le Pape, and Wim Nuijten, *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, vol. 39, Springer Science & Business Media, 2012.

[Bor12] Stephen P. Borgatti, *Social Network Analysis, Two-Mode Concepts in*, Computational Complexity, Springer, 2012, pp. 2912–2924.

[BP13] Simon Busard and Charles Pecheur, *PyNuSMV: NuSMV as a Python Library*, Nasa Formal Methods, Springer, 2013, pp. 453–458.

[BSR10] Roman Barták, Miguel A. Salido, and Francesca Rossi, *New Trends in Constraint Satisfaction, Planning, and Scheduling: A Survey*, The Knowledge Engineering Review **25** (2010), no. 03, 249–279.

[BSST09] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli, *Satisfiability Modulo Theories*, Handbook of Satisfiability **185** (2009), 825–885.

[BY04] Johan Bengtsson and Wang Yi, *Timed Automata: Semantics, Algorithms and Tools*, Lecture Notes in Computer Science **3098** (2004), 87–124.

[CC97] Sérgio Campos and Edmund Clarke, *The Verus Language: Representing Time Efficiently With BDDs*, Transformation-Based Reactive Systems Development, Springer, 1997, pp. 64–78.

[CCGR00] Alessandro Cimatti, Edmund Clarke, Fausto Giunchiglia, and Marco Roveri, *NuSMV: A New Symbolic Model Checker*, International Journal on Software Tools for Technology Transfer **2** (2000), no. 4, 410–425.

[CDL10] Edmund Clarke, Alexandre Donzé, and Axel Legay, *On Simulation-Based Probabilistic Model Checking of Mixed-Analog Circuits*, Formal Methods in System Design **36** (2010), no. 2, 97–113.

[CEN01] CENELEC, *EN. 50128: Railway applications-Communication, Signaling and Processing Systems-Software for Railway Control and Protection Systems*, 2001.

[CEN03] CENELEC, *EN. 50129: Railway application–Communications, Signaling and Processing Systems–Safety Related Electronic Systems for Signaling*, 2003.

[CFL+08] Edmund M. Clarke, James R. Faeder, Christopher J. Langmead, Leonard A. Harris, Sumit Kumar Jha, and Axel Legay, *Statistical Model Checking in BioLab: Applications to the Automated Analysis of T-Cell Receptor Signaling Pathway*, Computational Methods in Systems Biology, Springer, 2008, pp. 231–250.

[CGD09] Francesco Corman, Rob M. P. Goverde, and Andrea D'Ariano, *Rescheduling Dense Train Traffic Over Complex Station Interlocking Areas*, Robust and Online Large-Scale Optimization, Springer, 2009, pp. 369–386.

[CGM+98a] Alessandro Cimatti, Fausto Giunchiglia, Giorgio Mongardi, Dario Romano, Fernando Torielli, and Paolo Traverso, *Formal Verification of a Railway Interlocking System using Model Checking*, Formal Aspects of Computing **10** (1998), no. 4, 361–380.

[CGM+98b] Alessandro Cimatti, Fausto Giunchiglia, Giorgio Mongardi, Dario Romano, Fernando Torielli, and Paolo Traverso, *Model Checking Safety Critical Software with SPIN: an Application to a Railway Interlocking System*, Computer Safety, Reliability and Security (1998), 284–293.

[Che52] Herman Chernoff, *A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations*, The Annals of Mathematical Statistics (1952), 493–507.

[CHK+14] Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar, *An Overview of Recovery Models and Algorithms for Real-Time Railway Rescheduling*, Transportation Research Part B: Methodological **63** (2014), 15–37.

[Cho56] Noam Chomsky, *Three Models for the Description of Language*, Information Theory, IRE Transactions on **2** (1956), no. 3, 113–124.

[CKNZ12] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani, *Model Checking and the State Explosion Problem*, Tools for Practical Software Verification, Springer, 2012, pp. 1–30.

[CLS+17] Quentin Cappart, Christophe Limbrée, Pierre Schaus, Jean Quilbeuf, Louis-Marie Traonouez, and Axel Legay, *Verification of Interlocking Systems using Statistical Model Checking*, 18th International Symposium on High Assurance Systems Engineering (HASE), IEEE, 2017, pp. 61–68.

[CLSL15] Quentin Cappart, Christophe Limbrée, Pierre Schaus, and Axel Legay, *Verification by Discrete Simulation of Interlocking Systems*, 29th Annual European Simulation and Modelling Conference (ESM), 2015, pp. 402–409.

[Cor09] Thomas H. Cormen, *Introduction to Algorithms*, MIT press, 2009.

[Cri87] A. H. Cribbens, *Solid-State Interlocking (SSI): An Integrated Electronic Signalling System for Mainline Railways*, IEE Proceedings B (Electric Power Applications), vol. 134, IET, 1987, pp. 148–158.

[CS16] Quentin Cappart and Pierre Schaus, *A Dedicated Algorithm for Verification of Interlocking Systems*, International Conference on Computer Safety, Reliability, and Security, Springer, 2016, pp. 76–87.

[CS17] Quentin Cappart and Pierre Schaus, *Rescheduling Railway Traffic on Real Time Situations using Time-Interval Variables*, International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer, 2017, pp. 312–327.

[DCPP08] Andrea D'Ariano, Francesco Corman, Dario Pacciarelli, and Marco Pranzo, *Reordering and Local Rerouting Strategies to Manage Train Traffic in Real Time*, Transportation Science **42** (2008), no. 4, 405–419.

[DHK+14] Twan Dollevoet, Dennis Huisman, Leo Kroon, Marie Schmidt, and Anita Schobel, *Delay Management Including Capacities of Stations*, Transportation Science **49** (2014), no. 2, 185–203.

[DHSS12] Twan Dollevoet, Dennis Huisman, Marie Schmidt, and Anita Schobel, *Delay Management with Rerouting of Passengers*, Transportation Science **46** (2012), no. 1, 74–89.

[DMP91] Rina Dechter, Itay Meiri, and Judea Pearl, *Temporal Constraint Networks*, Artificial intelligence **49** (1991), no. 1-3, 61–95.

[DPP07] Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo, *A Branch and Bound Algorithm for Scheduling Trains in a Railway Network*, European Journal of Operational Research **183** (2007), no. 2, 643–657.

[DVCS15] Cyrille Dejemeppe, Sascha Van Cauwelaert, and Pierre Schaus, *The Unary Resource with Transition Times*, International Conference on Principles and Practice of Constraint Programming, Springer, 2015, pp. 89–104.

[Ear70] Jay Earley, *An Efficient Context-Free Parsing Algorithm*, Communications of the ACM **13** (1970), no. 2, 94–102.

[Eis99] Cindy Eisner, *Using Symbolic Model Checking to Verify the Railway Stations of Hoorn-Kersenboogerd and Heerhugowaard*, Correct Hardware Design and Verification Methods, Springer, 1999, pp. 99–109.

[EL86] Jürgen Eichenauer and Jürgen Lehn, *A Non-Linear Congruential Pseudo Random Number Generator*, Statistische Hefte **27** (1986), no. 1, 315–326.

[EO+98] James H. Earle, Denise Olsen, et al., *Engineering Design Graphics: AutoCAD Release 14*, Addison-Wesley Longman Publishing Co., Inc., 1998.

[Fay00] Alexander Fay, *A Fuzzy Knowledge-Based System for Railway Traffic Control*, Engineering Applications of Artificial Intelligence **13** (2000), no. 6, 719–729.

[FFdS+15] Marcelo Moretti Fioroni, Luiz Augusto G Franzese, Isac Reis de Santana, Pavel Emmanuel Pereira Lelis, Camila Batista da Silva, Gustavo Dezem Telles, José Alexandre Sereno Quintáns, Fábio Kikuda Maeda, and Rafael Varani, *From Farm to Port: Simulation of the Grain Logistics in Brazil*, Proceedings of the 2015 Winter Simulation Conference, IEEE Press, 2015, pp. 1936–1947.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos, *On Power-Law Relationships of the Internet Topology*, ACM SIGCOMM Computer Communication Review, vol. 29, ACM, 1999, pp. 251–262.

[FFM13] Alessandro Fantechi, Wan Fokkink, and Angelo Morzenti, *Some Trends in Formal Methods Applications to Railway Signaling*, Formal Methods for Industrial Critical Systems: A Survey of Applications (2013), 61–84.

[FG05] Cormac Flanagan and Patrice Godefroid, *Dynamic Partial-Order Reduction for Model Checking Software*, ACM Sigplan Notices, vol. 40, ACM, 2005, pp. 110–121.

[FLM+14] S. Foglietta, G. Leo, C. Mannino, P. Perticaroli, and M. Piacentini, *An Optimized, Automatic TMS in Operations in Roma Tiburtina and Monfalcone Stations*, WIT Transactions on The Built Environment **135** (2014), 635–647.

[For02] Bryan Ford, *Packrat Parsing: Simple, Powerful, Lazy, Linear Time, Functional Pearl*, ACM SIGPLAN Notices, vol. 37, ACM, 2002, pp. 36–47.

[Fre97] Eugene C. Freuder, *In Pursuit of the Holy Grail*, Constraints **2** (1997), no. 1, 57–61.

[FSK10] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno, *Cryptography Engineering*, John Wiley & Sons, 2010.

[Gen06] James E. Gentle, *Random Number Generation and Monte Carlo Methods*, Springer Science & Business Media, 2006.

[GHS15a] Steven Gay, Renaud Hartert, and Pierre Schaus, *Simple and Scalable Time-Table Filtering for the Cumulative Constraint*, International Conference on Principles and Practice of Constraint Programming, Springer, 2015, pp. 149–157.

[GHS15b] Steven Gay, Renaud Hartert, and Pierre Schaus, *Time-Table Disjunctive Reasoning for the Cumulative Constraint*, International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems, Springer, 2015, pp. 157–172.

[Gin66] Seymour Ginsburg, *The Mathematical Theory of Context Free Languages*, McGraw-Hill Book Company, 1966.

[GR14] George Raymond, *Where Are the CENELEC Standards Going ?*, IRSE News **Issue 203** (2014), 21–23.

[GS05] Radu Grosu and Scott A. Smolka, *Monte Carlo Model Checking*, Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2005, pp. 271–286.

[GSA04] Keivan Ghoseiri, Ferenc Szidarovszky, and Mohammad Jawad Asgharpour, *A Multi-Objective Train Scheduling Model and Solution*, Transportation research part B: Methodological **38** (2004), no. 10, 927–952.

[Gus88] John L. Gustafson, *Reevaluating Amdahl's Law*, Communications of the ACM **31** (1988), no. 5, 532–533.

[GW99] Ian P. Gent and Toby Walsh, *CSPLib: A Benchmark Library for Constraints*, International Conference on Principles and Practice of Constraint Programming, Springer, 1999, pp. 480–481.

[HA11] Alain Hait and Christian Artigues, *A Hybrid CP/MILP Method for Scheduling with Energy Costs*, European Journal of Industrial Engineering **5** (2011), no. 4, 471–489.

[Hei95] Philip Heidelberger, *Fast Simulation of Rare Events in Queueing and Reliability Models*, ACM Transactions on Modeling and Computer Simulation (TOMACS) **5** (1995), no. 1, 43–85.

[HGCC+98] Vicky Hartonas-Garmhausen, Sergio Campos, Alessandro Cimatti, Edmund Clarke, and Fausto Giunchiglia, *Verification of a Safety-Critical Railway Interlocking System with Real-Time Constraints*, Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, IEEE, 1998, pp. 458–463.

[HK02] Michael Huber and Steve King, *Towards an Integrated Model Checker for Railway Signalling Data*, FME 2002: Formal Methods - Getting IT Right, Springer, 2002, pp. 204–223.

[HKF96] Andrew Higgins, Erhan Kozan, and Luis Ferreira, *Optimal Scheduling of Trains on a Single Line Track*, Transportation Research Part B: Methodological **30** (1996), no. 2, 147–161.

[HKF97] Andrew Higgins, Erhan Kozan, and Luis Ferreira, *Heuristic Techniques for Single Line Train Scheduling*, Journal of Heuristics **3** (1997), no. 1, 43–62.

[HKNP06] Andrew Hinton, Marta Kwiatkowska, Gethin Norman, and David Parker, *PRISM: A Tool for Automatic Verification of*

*Probabilistic Systems*, Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2006, pp. 441–444.

[HNR16] Anne Elisabeth Haxthausen, Hoang Nga Nguyen, and Markus Roggenbach, *Comparing Formal Verification Approaches of Interlocking Systems*, International Conference on Reliability, Safety and Security of Railway Systems, Springer, 2016, pp. 160–177.

[HØ16] Anne E. Haxthausen and Peter H. Østergaard, *On the Use of Static Checking in the Verification of Interlocking Systems*, International Symposium on Leveraging Applications of Formal Methods, Springer, 2016, pp. 266–278.

[Hol97] Gerard J. Holzmann, *The model checker SPIN*, IEEE Transactions on software engineering **23** (1997), no. 5, 279–295.

[Hol06] G. Holzmann, *The Design of a Distributed Model Checking Algorithm for SPIN*, 2006.

[HPP13] Anne E. Haxthausen, Jan Peleska, and Ralf Pinger, *Applied Bounded Model Checking for Interlocking System Designs*, Software Engineering and Formal Methods, Springer, 2013, pp. 205–220.

[HR04] Michael Huth and Mark Ryan, *Logic in Computer Science: Modelling and Reasoning About Systems*, Cambridge University Press, 2004.

[HS16] Daniel Harabor and Peter J. Stuckey, *Rail Capacity Modelling with Constraint Programming*, International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems, Springer, 2016, pp. 170–186.

[Hut92] Graham Hutton, *Higher-Order Functions for Parsing*, J. Funct. Program. **2** (1992), no. 3, 323–343.

[JBJ08] Mark F. Jentsch, AbuBakr S. Bahaj, and Patrick A.B. James, *Climate Change Future Proofing of Buildings - Generation and Assessment of Building Simulation Weather Files*, Energy and Buildings **40** (2008), no. 12, 2148–2168.

[JCL+09] Sumit K. Jha, Edmund M. Clarke, Christopher J. Langmead, Axel Legay, André Platzer, and Paolo Zuliani, *A Bayesian Approach to Model Checking Biological Systems*, Computational Methods in Systems Biology, Springer, 2009, pp. 218–234.

[JLS12] Cyrille Jegourel, Axel Legay, and Sean Sedwards, *A Platform for High Performance Statistical Model Checking–PLASMA*, Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2012, pp. 498–503.

[JLS13] Cyrille Jegourel, Axel Legay, and Sean Sedwards, *Importance Splitting for Statistical Model Checking Rare Properties*, Computer Aided Verification, Springer, 2013, pp. 576–591.

[JLS14] Cyrille Jegourel, Axel Legay, and Sean Sedwards, *An Effective Heuristic for Adaptive Importance Splitting in Statistical Model Checking*, Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications, Springer, 2014, pp. 143–159.

[JSL13] Xianfei Jin, Appa Iyer Sivakumar, and Sing Yong Lim, *A Simulation Based Analysis on Reducing Patient Waiting Time for Consultation in an Outpatient Eye Clinic*, Proceedings of the 2013 Winter Simulation Conference (WSC), IEEE, 2013, pp. 2192–2203.

[Kam12] Norihiro Kamide, *Bounded Linear-Time Temporal Logic: A Proof-Theoretic Investigation*, Annals of Pure and Applied Logic **163** (2012), no. 4, 439–466.

[Kar06] Steven T. Karris, *Introduction to Simulink with Engineering Applications*, 2006.

[KB14] Wen-Yang Ku and J. Christopher Beck, *Revisiting Off-the-shelf Mixed Integer Programming and Constraint Programming Models for Job Shop Scheduling*, 2014.

[KBK+12] Elena Kelareva, Sebastian Brand, Philip Kilby, Sylvie Thiébaux, Mark Wallace, et al., *CP and MIP Methods for Ship Scheduling with Time-Varying Draft*, ICAPS, 2012.

[Kel01] John R. Kelly, *Cryptographically Secure Pseudo Random Number Generator*, August 14 2001, US Patent 6,275,586.

[KLM+98] Robert Kurshan, Vladimir Levin, Marius Minea, Doron Peled, and Hüsnü Yenigün, *Static Partial Order Reduction*, Tools and Algorithms for the Construction and Analysis of Systems (1998), 345–357.

[Knu98] Donald Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2, Addison-Wesley, 1998.

[KTK13] Elena Kelareva, Kevin Tierney, and Philip Kilby, *CP Methods for Scheduling and Routing with Time-Dependent Task Costs*, International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems, Springer, 2013, pp. 111–127.

[KWG09] Vineet Kahlon, Chao Wang, and Aarti Gupta, *Monotonic Partial Order Reduction: an Optimal Symbolic Partial Order Reduction Technique*, CAV, vol. 9, Springer, 2009, pp. 398–413.

[Lab09] Philippe Laborie, *IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems*, International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems, Springer, 2009, pp. 148–162.

[Lan12] Kevin Lano, *The B Language and Method: A Guide to Practical Formal Development*, Springer Science & Business Media, 2012.

[Lap92] Gilbert Laporte, *The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms*, European Journal of Operational Research **59** (1992), no. 3, 345–358.

[LCPT16] Christophe Limbrée, Quentin Cappart, Charles Pecheur, and Stefano Tonetta, *Verification of Railway Interlocking - Compositional Approach with OCRA*, Proceedings of the First International Conference on Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification (RSSRail), 2016, pp. 134–149.

[LDB10] Axel Legay, Benoît Delahaye, and Saddek Bensalem, *Statistical Model Checking: An Overview*, Runtime Verification, Springer, 2010, pp. 122–135.

[LDL+08] Tak Lam, Jianxun Jason Ding, Jyh-Charn Liu, et al., *XML Document Parsing: Operational and Performance Characteristics*, Computer (2008), no. 9, 30–37.

[Liv06] Benjamin Livshits, *Improving Software Security with Precise Static and Runtime Analysis*, 2006.

[LL09] Martin Lange and Hans Leiß, *To CNF or not to CNF? An efficient yet Presentable Version of the CYK Algorithm*, Informatica Didactica **8** (2009), 2008–2010.

[LM15] Leonardo Lamorgese and Carlo Mannino, *An Exact Decomposition Approach for the Real-Time Train Dispatching Problem*, Operations Research **63** (2015), no. 1, 48–64.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi, *UPPAAL in a Nutshell*, International Journal on Software Tools for Technology Transfer (STTT) **1** (1997), no. 1, 134–152.

[LR08] Philippe Laborie and Jerome Rogerie, *Reasoning with Conditional Time-Intervals*, FLAIRS Conference, 2008, pp. 555–560.

[LRSV09] Philippe Laborie, Jerome Rogerie, Paul Shaw, and Petr Vilím, *Reasoning with Conditional Time-Intervals Part II: An Algebraical Model for Resources*, FLAIRS Conference, 2009.

[LW66] Eugene L. Lawler and David E. Wood, *Branch-and-Bound Methods: A Survey*, Operations research **14** (1966), no. 4, 699–719.

[MB02] W. Scott Means and Michael A. Bodie, *The Book of SAX*, 2002.

[McN02] Ian McNeil, *An Encyclopedia of the History of Technology*, Routledge, 2002.

[MFH16] Hugo D. Macedo, Alessandro Fantechi, and Anne E. Haxthausen, *Compositional Verification of Multi-Station Interlocking Systems*, International Symposium on Leveraging Applications of Formal Methods, Springer, 2016, pp. 279–293.

[MN98] Makoto Matsumoto and Takuji Nishimura, *Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator*, ACM Transactions on Modeling and Computer Simulation (TOMACS) **8** (1998), no. 1, 3–30.

[MNR+12a] F. Moler, Hoang Nga Nguyen, Markus Roggenbach, S. A. Schneider, and Helen Treharne, *Combining Event-Based and State-Based Modelling for Railway Verification*, 2012.

[MNR+12b] Faron Moller, Hoang Nga Nguyen, Markus Roggenbach, Steve Schneider, and Helen Treharne, *CSPB Modelling for Railway Verification: the Double Junction Case Study*, Proceedings of the 12th International Workshop on Automated Verification of Critical Systems, 2012.

[MNR+12c] Faron Moller, Hoang Nga Nguyen, Markus Roggenbach, Steve Schneider, and Helen Treharne, *Defining and Model Checking Abstractions of Complex Railway Models Using CSP‖B*, Haifa Verification Conference, Springer, 2012, pp. 193–208.

[MP02] Alessandro Mascis and Dario Pacciarelli, *Job-Shop Scheduling with Blocking and No-wait Constraints*, European Journal of Operational Research **143** (2002), no. 3, 498–517.

[MPO08] Adriaan Moors, Frank Piessens, and Martin Odersky, *Parser Combinators in Scala*, 2008.

[MPP05] Michael D. Moffitt, Bart Peintner, and Martha E. Pollack, *Augmenting Disjunctive Temporal Problems with Finite-Domain Constraints*, Proceedings of the National Conference on Artificial Intelligence, vol. 20, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005, p. 1187.

[MV03] Klaus Muller and Tony Vignaux, *Simpy: Simulating Systems in Python*, 2003.

[NH04] Andrew Nash and Daniel Huerlimann, *Railroad Simulation using OpenTrack*, Computers in Railways IX (2004), 45–54.

[NHSK04] Andrew Nash, Daniel Huerlimann, Jörg Schütte, and Vasco Paul Krauss, *RailML–A Standard Data Interface for Railroad Applications*, Computers in Railways IX, WIT Press, Southampton (2004), 233–240.

[OFWB03] Joshua O'Madadhain, Danyel Fisher, Scott White, and Y. Boey, *The Jung (Java Universal Network/Graph) Framework*, 2003.

[OIY+09] Simon Ostermann, Alexandria Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema, *A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing*, Cloud Computing, Springer, 2009, pp. 115–131.

[Osc12] OscaR Team, *OscaR: Scala in OR*, 2012, Available from `https://bitbucket.org/oscarlib/oscar`.

[Pel98] Doron Peled, *Ten Years of Partial Order Reduction*, Computer Aided Verification, Springer, 1998, pp. 17–28.

[Pin15] Michael Pinedo, *Scheduling*, Springer, 2015.

[Rei] Anthony J. Dos Reis, *Recursive-Descent Parsing*, Compiler Construction using Java, JavaCC, and Yacc, 185–214.

[RK11] Reuven Y. Rubinstein and Dirk P. Kroese, *Simulation and the Monte Carlo Method*, vol. 707, John Wiley & Sons, 2011.

[RN07] Manuel D. Rossetti and Shikha Nangia, *An Object-Oriented Framework for Simulating Full Truckload Transportation Networks*, Proceedings of the 39th Conference on Winter Simulation: 40 years! The Best is yet to Come, IEEE Press, 2007, pp. 1869–1877.

[Rob14] Stewart Robinson, *Simulation: the Practice of Model Development and Use*, Palgrave Macmillan, 2014.

[Rod07] Joaquín Rodriguez, *A Constraint Programming Model for Real-Time Train Scheduling at Junctions*, Transportation Research Part B: Methodological **41** (2007), no. 2, 231–245.

[RVBW06] Francesca Rossi, Peter Van Beek, and Toby Walsh, *Handbook of Constraint Programming*, Elsevier, 2006.

[S. 83] S. Araya and K. Abe and K. Fukumori, *An Optimal Rescheduling for Online Train Traffic Control in Disturbed Situations*, The 22nd IEEE Conference on Decision and Control, Dec 1983, pp. 489–494.

[SBS11] Samuel Sogin, Christopher P. L. Barkan, and Mohd Rapik Saat, *Simulating the Effects of Higher Speed Passenger Trains in Single Track Freight Networks*, Proceedings of the Winter Simulation Conference, Winter Simulation Conference, 2011, pp. 3684–3692.

[SCdB15] Pengfei Sun, Simon Collart-dutilleul, and Philippe Bon, *A Model Pattern of Railway Interlocking System by Petri Nets*, Models and Technologies for Intelligent Transportation Systems (MT-ITS), 2015 International Conference on, IEEE, 2015, pp. 442–449.

[Sco98] Roger S. Scowen, *Extended BNF-A Generic Base Standard*, Tech. report, 1998.

[SEB+13] Miguel A. Salido, Joan Escamilla, Federico Barber, Adriana Giret, Dunbing Tang, and Min Dai, *Energy-Aware Parameters in Job-Shop Scheduling Problems*, GREEN-COPLAS 2013: IJCAI 2013 Workshop on Constraint Reasoning, Planning and Scheduling Problems for a Sustainable Future, 2013, pp. 44–53.

[Sha94] Perwez Shahabuddin, *Importance Sampling for the Simulation of Highly Reliable Markovian Systems*, Management Science **40** (1994), no. 3, 333–352.

[SS10] Michael Schachtebeck and Anita Schobel, *To Wait or not to Wait-and Who Goes First ? Delay Management with Priority Decisions*, Transportation Science **44** (2010), no. 3, 307–321.

[Str01] Steven H. Strogatz, *Exploring Complex Networks*, Nature **410** (2001), no. 6825, 268–276.

[SVA05] Koushik Sen, Mahesh Viswanathan, and Gul Agha, *On Statistical Model Checking of Stochastic Systems*, Computer Aided Verification, Springer, 2005, pp. 266–280.

[SWDK15] Ilankaikone Senthooran, Mark Wallace, and Leslie De Koninck, *Freight Train Threading with Different Algorithms*, International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems, Springer, 2015, pp. 393–409.

[SYB04] Anthony Sulistio, Chee Shin Yeo, and Rajkumar Buyya, *A Taxonomy of Computer-Based Simulations and its Mapping to Parallel and Distributed Systems Simulation Tools*, Software-Practice and Experience **34** (2004), no. 7, 653–674.

[TAV09] G. Theeg, E. Anders, and S.V. Vlasenko, *Railway Signalling & Interlocking: International Compendium*, Eurailpress, 2009.

[TR95] Robert C. Thomson and Dianne E. Richardson, *A Graph Theory Approach to Road Network Generalisation*, Proceeding of the 17th International Cartographic Conference, 1995, pp. 1871–1880.

[TRN+02] David Tombs, Neil Robinson, George Nikandros, et al., *Signalling Control Table Generation and Verification*, CORE 2002: Cost Efficient Railways through Engineering (2002), 415.

[VA10] Somsak Vanit-Anunchai, *Modelling Railway Interlocking Tables using Coloured Petri Nets*, Coordination Models and Languages, Springer, 2010, pp. 137–151.

[VCDMS16] Sascha Van Cauwelaert, Cyrille Dejemeppe, Jean-Noël Monette, and Pierre Schaus, *Efficient Filtering for the Unary Resource with Family-Based Transition Times*, International Conference on Principles and Practice of Constraint Programming, Springer, 2016, pp. 520–535.

[VHM09] Pascal Van Hentenryck and Laurent Michel, *Constraint-Based Local Search*, MIT press, 2009.

[VHP14] Linh H. Vu, Anne Elisabeth Haxthausen, and Jan Peleska, *Formal Modeling and Verification of Interlocking Systems Featuring Sequential Release*, Formal Techniques for Safety-Critical Systems, Springer, 2014, pp. 223–238.

[Vil04] Petr Vilím, *O(n log n) Filtering Algorithms for Unary Resource Constraint*, International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming, Springer, 2004, pp. 335–347.

[Vil07] Petr Vilím, *Global Constraints in Scheduling*, Ph.D. thesis, PhD Thesis, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Mathematical Logic, KTIML MFF, Universita Karlova, Malostranské námestı 2/25, 118 00 Praha 1, Czech Republic, 2007.

[Vil09] Petr Vilím, *Max Energy Filtering Algorithm for Discrete Cumulative Resources*, International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems, Springer, 2009, pp. 294–308.

[VLS15] Petr Vilím, Philippe Laborie, and Paul Shaw, *Failure-directed Search for Constraint-Based Scheduling*, International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems, Springer, 2015, pp. 437–453.

[W+01] Douglas Brent West et al., *Introduction to Graph Theory*, vol. 2, Prentice Hall Upper Saddle River, 2001.

[WH82] Brian A. Wichmann and I. David Hill, *Algorithm AS 183: An Efficient and Portable Pseudo-Random Number Generator*, Journal of the Royal Statistical Society. Series C (Applied Statistics) **31** (1982), no. 2, 188–190.

[Win02] Kirsten Winter, *Model Checking Railway Interlocking Systems*, Australian Computer Science Communications, vol. 24, Australian Computer Society, Inc., 2002, pp. 303–310.

[Win12] Kirsten Winter, *Optimising Ordering Strategies for Symbolic Model Checking of Railway Interlockings*, Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies, Springer, 2012, pp. 246–260.

[WJR+06] K. Winter, W. Johnston, P. Robinson, P. Strooper, and L. Van Den Berg, *Tool Support for Checking Railway Interlocking Designs*, Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software, vol. 55, Australian Computer Society, Inc., 2006, pp. 101–107.

[Won91] Wai Wong, *A Simple Graph Theory and its Application in Railway Signaling*, International Workshop on the HOL Theorem Proving System and Its Applications, IEEE, 1991, pp. 395–409.

[WUS+10] Travis Worth, Reha Uzsoy, Erika Samoff, Anne-Marie Meyer, Jean-Marie Maillard, and Aaron M. Wendelboe, *Modelling the Response of a Public Health Department to Infectious Disease*, Proceedings of the 2010 Winter Simulation Conference (WSC), IEEE, 2010, pp. 2185–2198.

[XTGC09] Tianhua Xu, Tao Tang, Chunhai Gao, and Baigen Cai, *Logic Verification of Collision Avoidance System in Train Control Systems*, Intelligent Vehicles Symposium, IEEE, 2009, pp. 918–923.

[YN97] Takeshi Yamada and Ryohei Nakano, *Job Shop Scheduling*, IEE Control Engineering Series (1997), 134–134.

[ZZ05] Xuesong Zhou and Ming Zhong, *Bicriteria Train Scheduling for High-Speed Passenger Railroad Planning Applications*, European Journal of Operational Research **167** (2005), no. 3, 752–771.