Journal of Computational and Applied Mathematics 466 (2025) 116594

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

A library in CoCoA for implementing railway interlocking systems

Antonio Hernando [a], José Luis Galán–García [b,*], Yolanda Padilla–Domínguez [b], María Ángeles Galán–García [b], Gabriel Aguilera–Venegas [b]

[a] Departamento de Sistemas Informáticos, E.T.S.I. de Sistemas Informáticos, Universidad Politécnica de Madrid, Madrid, Spain
[b] Departamento de Matemática Aplicada, Escuela de Ingenierías Industriales, Universidad de Málaga, Málaga, Spain

ARTICLE INFO

Keywords:
Railway interlocking system
Computer algebra
Decision making
Commutative algebra

ABSTRACT

In this paper, we propose a user-friendly library in CoCoA to address and completely resolve the challenges posed by the highly efficient and intriguing mathematical model introduced in Hernando et al., (2023) for implementing railway interlocking systems. Although the algebraic model (Hernando et al., 2023) allows for fast performance, it requires implementers and users to have a high level of mathematical knowledge, mastering concepts such as Gröbner bases, ideals, rings, and polynomials. This expertise is necessary to manually define ideals generated by numerous complex polynomials in multiple variables, which depend on the railway station's topology, a process that can be both tedious and error-prone. To completely resolve these challenges, we have developed a CoCoA library that streamlines the implementation of interlocking systems using our mathematical framework, effectively eliminating manual errors. Consequently, thanks to the library we have developed and presented here, even users without mathematical knowledge can easily implement and manage a railway interlocking system.

1. Introduction

Globally, rail transportation serves as a pivotal conduit, facilitating the reliable and efficient movement of goods and individuals across extensive distances. Its significance in bolstering the economy and societal structures is underscored in numerous resilience-focused studies on railway transport systems [1,2].

At the heart of rail transportation lies the railway interlocking system, a safety-critical construct designed to avert train collisions. Ensuring the compatibility of switch positions and signal indications at a railway station is paramount for safety. Interlocking thwarts inappropriate alterations to traffic signals and turnout switches. A railway station consists of sections linked by traffic signals and turnouts, delineating potential train movements. A route is a series of interconnected sections that a train can traverse, divided into several blocks. To avoid collisions, two trains must never occupy the same block of a route simultaneously. Furthermore, two intersecting routes (or their relevant blocks) must not be allocated to trains at the same time.

As mentioned in [3], the safety technology in railway interlocking systems has significantly evolved in Europe, transitioning from mechanical devices to computerized systems. The evolution of railway interlocking systems has been a significant focus in recent years, with advancements transitioning from mechanical devices to sophisticated computerized systems. This transformation is crucial for enhancing the safety and efficiency of railway operations. As [1] highlights, resilience in railway transport systems is paramount, and modern interlocking systems play a vital role in achieving this resilience by ensuring safe and reliable train movements. Moreover, the historical context provided by [4] underscores the importance of regional railways in economic

* Corresponding author.
E-mail addresses: antonio.hernando@upm.es (A. Hernando), jlgalan@uma.es (J.L. Galán–García), ypadilla@ctima.uma.es (Y. Padilla–Domínguez), magalan@ctima.uma.es (M.Á. Galán–García), gabri@ctima.uma.es (G. Aguilera–Venegas).

https://doi.org/10.1016/j.cam.2025.116594
Received 30 September 2024; Received in revised form 24 January 2025
Available online 24 February 2025

development. Their research indicates that the evolution of railway infrastructure, including interlocking systems, has been instrumental in shaping regional economies. This historical perspective is essential for understanding the long-term impact of technological advancements in railway systems. Recent studies, such as those by [5], emphasize the ongoing developments in interlocking technology. Huang's work provides a comprehensive overview of the past, present, and future of railway interlocking systems, highlighting the continuous improvements and innovations that drive the industry forward. Additionally, [6] discuss the analysis of relay interlocking systems using SMT-based model checking, which offers a modern approach to ensuring the reliability and safety of these systems.

This intricate problem has been the subject of extensive research with recent studies delving into artificial intelligence applications for fault detection [7] and contrasting different safety verification methods [8–10]. Efforts have also been made towards developing formal model-based methodologies to assist railway engineers in specifying and verifying interlocking systems [11].

The authors have dedicated several years to studying this problem, resulting in the development of various algebraic models. Some of these models draw on polynomials, ideals, and Gröbner bases, akin to those utilized in Artificial Intelligence for implementing expert systems [12,13]. This approach forms a bridge between computational algebra and interlocking problems, suggesting that computer algebra systems can be employed to implement interlocking systems. A groundbreaking algebraic model for implementing interlocking systems was recently introduced [14], from which several extensions have been developed [15,16]. This model offers a linear algorithm that significantly surpasses previous models in performance, making it ideal for large-scale railway stations. The algorithm utilizes the representation of the railway station through polynomials. With this representation, the task of identifying dangerous situations in a railway station is transformed into the calculation of the residue of a monomial division over a set of polynomials.

Nevertheless, despite the speed and linearity of this mathematical framework, which gives it a significant edge over others, it presents several drawbacks that prevent it from being applied in practice.

- Implementing an interlocking system for a railway station based on this algebraic method [14] is laborious and complex. Only mathematicians with a strong background in algebra, polynomial division, and Gröbner bases can carry out this implementation.
- Both implementing and managing the interlocking system according to this model require handling a vast number of unintuitive polynomials in many variables. This task can be extremely labor-intensive and prone to errors, even for expert mathematicians.

In response to this, in this paper we introduce a library in *CoCoA* [17]. This library streamlines the implementation of interlocking systems based on our mathematical framework. Consequently, it enables users, even those without any mathematical background, to implement and manage a railway interlocking system with ease and simplicity.

The paper is organized as follows. In Section 2, we provide a broad overview of the mathematical framework presented in [14] and discuss the driving forces behind our proposal. To illustrate our motivation we present an example in Section 3. Section 4 is dedicated to detailing the architecture of our proposed solution. In Section 5, we illustrate how our proposal streamlines the implementation and management of an interlocking system. The technical aspects of our library's implementation are thoroughly explored in Sections 6 and 7. Finally, we present our conclusions in Section 8. All files related to the library and the illustrative example used in this manuscript to help readers understand the concepts presented can be downloaded at the following URL: http://u.uma.es/fTY/. This link also includes files associated with a real Spanish railway station (Algodor Railway Station) to demonstrate the benefits of our library with a real-life example.

2. Motivation

The methodology outlined in [14] offers a mathematical framework for assessing whether a railway station's situation is dangerous. This approach views a railway station as a collection of sections and a relation, E, between these sections that signifies possible connections. In essence, E is a subset of section pairs, such that $(a, b) \in E$ if and only if a train can transition from section a to section b under a certain railway station configuration (e.g., the colour light signal indication or the state of the turnouts and crossovers switches). The set E encapsulates potential section connections and represents the static aspect of the railway station, independent of colour light signal indications or turnout and crossover switch positions.

The actual connections within the railway station are governed by the colour light signals and the state of the turnouts and crossovers. As per [14], the set $P \in E$ denotes the real connections in the station for a specific configuration. This set P alters when there are changes in the colour light signals or the state of the turnouts and crossovers. Along with the set P of section pairs, a multiset Q is also defined to indicate train positions. The set P and multiset Q constitute the dynamic aspect of the railway station, defining a situation that can change, and whose safety we aim to determine.

The approach in [14] translate all of these aspects into algebraic terms.

Static Aspect. It defines a polynomial ring is several variables, $\mathcal{A}$, and a list $\mathcal{E}$ of polynomials in this ring $\mathcal{A}$. The diagram depicted in Fig. 1 provides a visual representation of this process.

Dynamic Aspect. For each situation in the railway station, it defines two monomials p and q in $\mathcal{A}$ that represent the set P and the multiset Q. If trains in the railway station occupy one section, we can determine if the situation is dangerous by checking if the $\mathrm{NR}(pq, \mathcal{E})$, where NR is the remainder of dividing a polynomial over a set of polynomials. In the event that a train

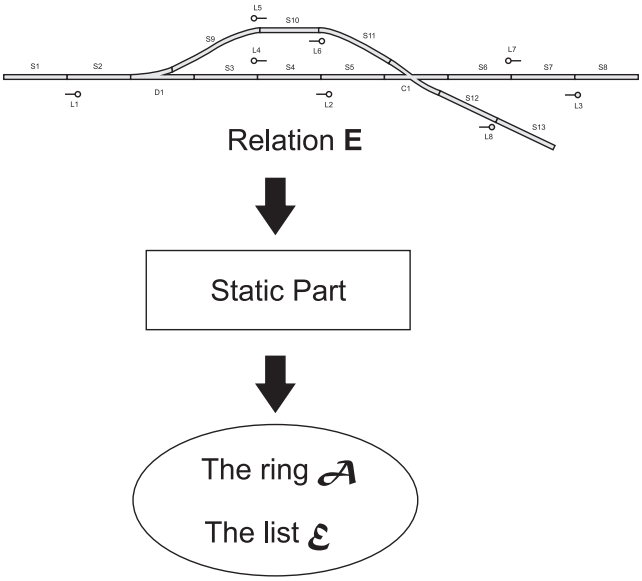

Fig. 1. Static Part.

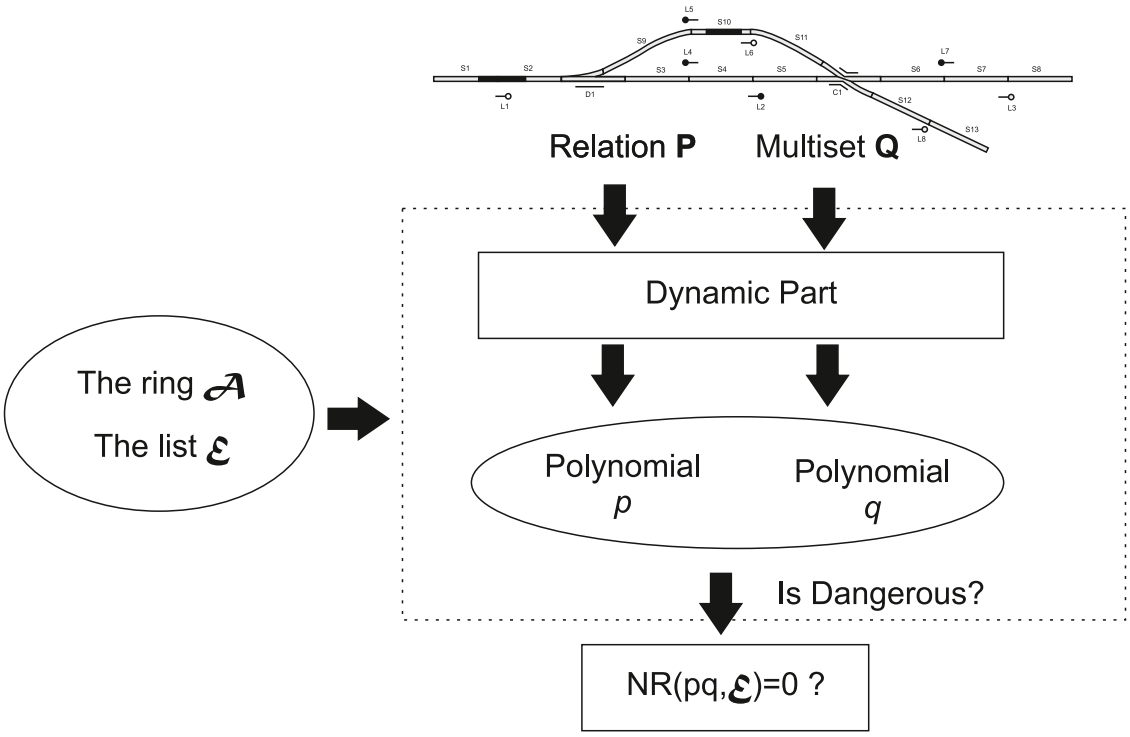

Fig. 2. Dynamic Part.

occupies multiple sections, the approach in [14] converts the situation into an equivalent one where trains are positioned in single sections. This transformation entails modifications to the previously defined monomials p and q. The diagram depicted in Fig. 2 provides a visual representation of this process. This dynamic aspect is what it is called the interlocking system of the specific railway station.

According to the approach in [14], an interlocking system can be implemented in a Computer Algebra System to ascertain whether a situation is dangerous. However, this type of implementation has significant drawbacks for practical use:

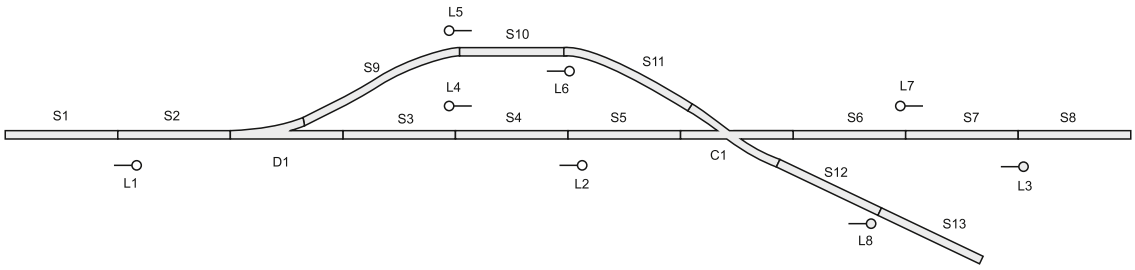

Fig. 3. A railway station.

- The implementation of an interlocking system for a railway station is quite laborious and not straightforward. The set E is dependent on the semaphores, turnouts, and crossovers. It would be beneficial to have methods that generate the set E by specifying the semaphores, turnouts, and crossovers in the railway station.
- This implementation can only be carried out by a mathematician with a substantial background in algebra, polynomial division, and Gröbner basis. Even with this background, the process for obtaining the ring $\mathcal{A}$ and the list $\mathcal{E}$ is very tedious.
- The monomial p is not readily updated when the state of colour light signals and the switch of turnouts and crossovers change. Furthermore, when a train occupies several sections simultaneously, the monomials p and q need to be updated, and obtaining them is not straightforward. Therefore, when the situation in a railway station changes, it necessitates the updating of both monomials p and q. Having automated methods to facilitate this would be advantageous.

In order to overcome all of these drawbacks, we have developed a library in *CoCoA* that offers the user operations and functions to implement an interlocking system for any railway station.

3. Example

To illustrate the above and particularly highlight the advantages of our library, we present an example of a small railway station (see Fig. 3). In this railway station, we compare two implementations: one without our library (see Section 3.1) and one with it (see Section 3.2). As will be seen, the implementation without the library requires a significant background in algebra and involves manually defining numerous variables and intricate polynomials, as specified by the model [14], to design an interlocking system—a very non-intuitive task prone to numerous errors. In contrast, the implementation using our library requires minimal, intuitive, and error-free instructions.

The main aim here is to illustrate this comparison, highlighting its complexity and non-intuitive nature for implementation without the library, rather than delving into the computational details of the mathematical aspects (described in [14]). It particularly stands out when compared to this very same railway station implemented using the library we propose here. The reader will notice how effortless it becomes.

We will examine the railway station shown in Fig. 3. While this simple illustrative example helps readers understand the concepts presented, we have also included a real Spanish railway example (Algodor Railway Station) at http://u.uma.es/fTY/ (see files `Algodor.cocoa5` and `Algodor source.txt`). This allows readers to download the station and interact with it, engaging more closely with our research.

The railway station in Fig. 3 comprises 13 sections (S1 to S8), 8 semaphores (L1 to L8), one turnout (D1), and one crossover (C1). Fig. 3 depicts the static aspects discussed earlier, as it does not specify the colour of the semaphores, the state of the turnout and crossover switch positions, or the positions of the trains.

On the other hand, Fig. 4 illustrates the dynamic aspect discussed above, that is, different situations within the station, specifying the trains located in the station, the colour of the semaphores, and the position of the switches for the turnout and crossover in a defined state. Specifically, Fig. 4 illustrates five different situations: A, B, C, D, and E. In this figure, the colour of the light signal is denoted by circles filled in black for red and white for green. As can be observed, in scenario A, the light signal between sections S1 and S2 is green, while the signal between sections S7 and S6 is red. Similarly, we describe the position of the switch for both the turnout and the crossover. It can be seen that in scenario A, the switch of the turnout between sections S2, S9, and S3 is in a straight position, whereas the switch of the crossover between sections S5, S6, S11, and S12 is diverted.

We consider different situations derived from Scenario A. In Scenario B in Fig. 4, we illustrate the situation where the train previously in section S1 has moved to section S2. In Scenario C, we explore the situation where a new train appears in section S8. In Scenario D, the light signal L6 has been adjusted to display red. Finally, in Scenario E, we examine the situation where the light signal L7 has been set to green and the switch of the crossover C1 has been set to the straight track position.

3.1. Implementation without the library

Here we present the implementation of the interlocking system for the previous railway station using the model described in [14], without utilizing the library. We will not explain or detail the model from [14], as our aim is solely to illustrate that

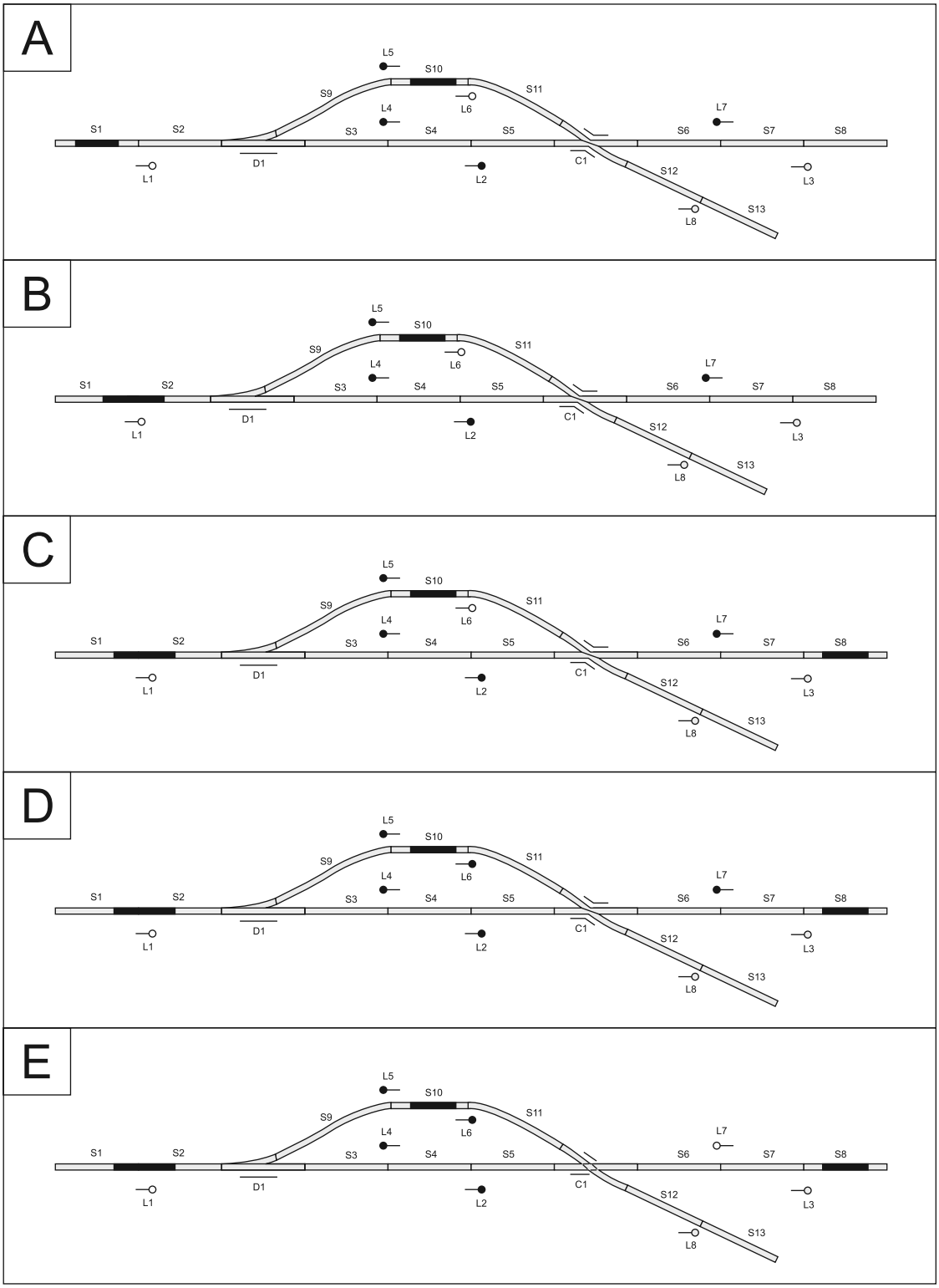

Fig. 4. Dynamic part.

this model requires the implementer to have a strong mathematical background and mastery of concepts such as rings, ideals, polynomials in several variables, and normal forms. Additionally, as the reader will appreciate, even for this very simple railway station, the implementer needs to define a polynomial ring with a large number of variables and generate ideals from numerous abstract polynomials. We believe this example gives the reader an idea that if such a small station requires this many polynomials and variables, the amount needed for a much larger, real-world station would be significantly greater. Therefore, without our library, implementing an interlocking system based on this algebraic model for a large railway station can become nearly impossible, or at the very least, exceedingly tedious and challenging.

We will consider the both the static and dynamic aspect of the interlocking system:

Static Aspect. For the railway in Fig. 3 the set E is:

$\{(1,2),(2,1),(4,5),(5,4),(7,8),(8,7),(4,3),(3,4),(10,9),(9,10),$
$(10,11),(11,10),(7,6),(6,7),(12,13),(13,12),(2,3),(2,9),(3,2),$
$(9,2),(5,6),(5,12),(11,12),(11,6),(6,5),(12,5),(12,11),(6,11)\}$

As can be observed, sections S4 and S5 are interconnected, hence $(4,5) \in E$. If the light signal L2 displays a green signal it allows for a train to transition from section S4 to section S5. The static aspect of the railway station is defined by means of the polynomial ring $\mathcal{A}$ and a list $\mathcal{E}$. In the case of the railway station in Fig. 3 we have that:

- The ring $\mathcal{A}$ is:

$$\mathcal{A} = \mathbb{Z}_2[l_1 \dots l_{28}, m_1, \dots, m_{28}, t_1, \dots, t_{13}]$$

  There exists a one-to-one correspondence between the edges in E and the variables of types l_i and m_i. There are the variables $l_1, \dots, l_{28}$ and $m_1, \dots, m_{28}$, which align with the edges $(1,2), \dots (6,11)$. In a similar fashion, another bijection is formed between the sections and variables of type t_i. With the presence of 13 sections, we have an equivalent number of variables of type t_i, represented as $t_1, \dots, t_{13}$, corresponding to sections S1 through S13.
- The list $\mathcal{E}$ is:

$[l_1l_2t_1 + m_1m_2t_1t_2, l_1m_2t_1 + m_1m_2t_1t_2, l_1l_2t_2 + m_1m_2t_1t_2, l_2m_1t_2 + m_1m_2t_1t_2, l_3l_4t_4 + m_3m_4t_4t_5,$
$l_3m_4t_4 + m_3m_4t_4t_5, l_3l_4t_5 + m_3m_4t_4t_5, l_4m_3t_5 + m_3m_4t_4t_5, l_5l_6t_7 + m_5m_6t_7t_8, l_5m_6t_7 + m_5m_6t_7t_8,$
$l_5l_6t_8 + m_5m_6t_7t_8, l_6m_5t_8 + m_5m_6t_7t_8, l_7l_8t_4 + m_7m_8t_3t_4, l_7m_8t_4 + m_7m_8t_3t_4, l_7l_8t_3 + m_7m_8t_3t_4,$
$l_8m_7t_3 + m_7m_8t_3t_4, l_9l_{10}t_{10} + m_9m_{10}t_9t_{10}, l_9m_{10}t_{10} + m_9m_{10}t_9t_{10}, l_9l_{10}t_9 + m_9m_{10}t_9t_{10},$
$l_{10}m_9t_9 + m_9m_{10}t_9t_{10}, l_{11}l_{12}t_{10} + m_{11}m_{12}t_{10}t_{11}, l_{11}m_{12}t_{10} + m_{11}m_{12}t_{10}t_{11}, l_{11}l_{12}t_{11} + m_{11}m_{12}t_{10}t_{11},$
$l_{12}m_{11}t_{11} + m_{11}m_{12}t_{10}t_{11}, l_{13}l_{14}t_7 + m_{13}m_{14}t_6t_7, l_{13}m_{14}t_7 + m_{13}m_{14}t_6t_7, l_{13}l_{14}t_6 + m_{13}m_{14}t_6t_7,$
$l_{14}m_{13}t_6 + m_{13}m_{14}t_6t_7, l_{15}l_{16}t_{12} + m_{15}m_{16}t_{12}t_{13}, l_{15}m_{16}t_{12} + m_{15}m_{16}t_{12}t_{13}, l_{15}l_{16}t_{13} + m_{15}m_{16}t_{12}t_{13},$
$l_{16}m_{15}t_{13} + m_{15}m_{16}t_{12}t_{13}, l_{17}l_{19}t_2 + m_{17}m_{19}t_2t_3, l_{17}m_{19}t_2 + m_{17}m_{19}t_2t_3, l_{18}l_{20}t_2 + m_{18}m_{20}t_2t_9,$
$l_{18}m_{20}t_2 + m_{18}m_{20}t_2t_9, l_{17}l_{19}t_3 + m_{17}m_{19}t_2t_3, l_{19}m_{17}t_3 + m_{17}m_{19}t_2t_3, l_{18}l_{20}t_9 + m_{18}m_{20}t_2t_9,$
$l_{20}m_{18}t_9 + m_{18}m_{20}t_2t_9, l_{21}l_{25}t_5 + m_{21}m_{25}t_5t_6, l_{21}m_{25}t_5 + m_{21}m_{25}t_5t_6, l_{22}l_{26}t_5 + m_{22}m_{26}t_5t_{12},$
$l_{22}m_{26}t_5 + m_{22}m_{26}t_5t_{12}, l_{23}l_{27}t_{11} + m_{23}m_{27}t_{11}t_{12}, l_{23}m_{27}t_{11} + m_{23}m_{27}t_{11}t_{12}, l_{24}l_{28}t_{11} + m_{24}m_{28}t_6t_{11},$
$l_{24}m_{28}t_{11} + m_{24}m_{28}t_6t_{11}, l_{21}l_{25}t_6 + m_{21}m_{25}t_5t_6, l_{25}m_{21}t_6 + m_{21}m_{25}t_5t_6, l_{22}l_{26}t_{12} + m_{22}m_{26}t_5t_{12},$
$l_{26}m_{22}t_{12} + m_{22}m_{26}t_5t_{12}, l_{23}l_{27}t_{12} + m_{23}m_{27}t_{11}t_{12}, l_{27}m_{23}t_{12} + m_{23}m_{27}t_{11}t_{12}, l_{24}l_{28}t_6 + m_{24}m_{28}t_6t_{11},$
$l_{28}m_{24}t_6 + m_{24}m_{28}t_6t_{11}, t_1^2, t_2^2, t_3^2, t_4^2, t_5^2, t_6^2, t_7^2, t_8^2, t_9^2, t_{10}^2, t_{11}^2, t_{12}^2, t_{13}^2, t_5t_{11}]$

As can be seen, the list $\mathcal{E}$ contains numerous highly intricate abstract polynomials for this simple railway station.

Dynamic Aspect. We will consider the different scenarios illustrated in Fig. 4.

- Scenario A. For scenario A, the polynomial p is:

$$l_1l_2l_4l_5l_6l_8l_{10}l_{11}l_{12}l_{14}l_{15}l_{16}l_{17}l_{19}l_{22}l_{24}l_{26}l_{28}m_3m_7m_9m_{13}m_{18}m_{20}m_{21}m_{23}m_{25}m_{27}$$

  For a given edge $(a,b) \in E$, the associated variable l_i is in p if and only if a train can pass from section a to section b. Conversely, the corresponding variable m_i is in p if and only if a train is unable to make the transition from section a to section b. As an example, scenario A in Fig. 4, the light signal between sections S1 and S2 is green, indicating that a train can pass from section S1 to section S2. Therefore, l_1, the variable of type l_i that corresponds to the edge $(1,2)$, is in the monomial p. On the other hand, the light signal between sections S7 and S6 is red in scenario A in Fig. 4, signifying that a train cannot pass from section S7 to section S6. As a result, m_{13}, the variable of type m_i corresponding to the edge $(7,6)$, is also included in the monomial p.
  The polynomial q for scenario A in Fig. 4 is:

$$t_1t_{10}$$

  A variable of the form t_i is in the monomial q if and only if the corresponding section S_i is occupied by a train. Given that sections S1 and S10 are currently occupied by trains, the monomial q is expressed as $q = t_1t_{10}$.
  To ascertain the safety of the situation, we need to calculate the following:

$$\mathrm{NR}(pq, \mathcal{E})$$

  This calculation can be efficiently performed using a Computer Algebra System. Given that the resulting value is non-zero, we can confidently conclude that the situation is indeed safe.

- Scenario B. Next, we will examine the scenario B depicted in Fig. 4. This involves the problem of a train occupying various sections.
  The framework in [14] does not initially account for trains that occupy multiple sections simultaneously. When a train occupies an edge (a, b), the framework first transforms the situation into an equivalent one where passage from section a to section b is no longer feasible. This transformation is achieved by updating the monomial p as follows:

  $$l_4l_5l_6l_8l_{10}l_{11}l_{12}l_{14}l_{15}l_{16}l_{17}l_{19}l_{22}l_{24}l_{26}l_{28}m_1m_2m_3m_7m_9m_{13}m_{18}m_{20}m_{21}m_{23}m_{25}m_{27}$$

  As can be observed in scenario B in Fig. 4, a train occupies the edges (1,2) and (2,1), and consequently, the variables l_1 and l_2 are excluded from the monomial p, while the variables m_1 and m_2 are introduced into p.
  The variable q also requires an update:

  $$t_1t_2t_{10}$$

  To verify the safety of the situation, we need to perform the following calculation:

  $$\mathrm{NR}(pq, \mathcal{E})$$

  As the resulting value continues to deviate from zero, we can maintain that the situation remains safe.
- Scenario C. This involves a new train in section S8. This requires an update to the variable q.

  $$t_1t_2t_8t_{10}$$

  To verify the safety of the situation, we need to perform the following calculation:

  $$\mathrm{NR}(pq, \mathcal{E})$$

  Given that the resulting value is zero, we can conclude that the situation is dangerous. As can be observed, there is a risk of collision between the trains in S10 and S8.
- Scenario D. In this scenario, the light signal L6 has been adjusted to display red. To reflect this change, we need to update the polynomial p:

  $$l_4l_5l_6l_8l_{10}l_{12}l_{14}l_{15}l_{16}l_{17}l_{19}l_{22}l_{24}l_{26}l_{28}m_1m_2m_3m_7m_9m_{11}m_{13}m_{18}m_{20}m_{21}m_{23}m_{25}m_{27}$$

  To verify the safety of the situation, we need to perform the following calculation:

  $$\mathrm{NR}(pq, \mathcal{E})$$

  Given that the resulting value is non-zero, we can conclude that the situation has returned to a state of safety.
- Scenario E. In this scenario, the light signal L7 has been set to green and the switch of the crossover C1 has been set to straight track position. To accommodate these changes, we have to modify the polynomial p:

  $$l_4l_5l_6l_8l_{10}l_{12}l_{13}l_{14}l_{15}l_{16}l_{17}l_{19}l_{21}l_{23}l_{25}l_{27}m_1m_2m_3m_7m_9m_{11}m_{18}m_{20}m_{22}m_{24}m_{26}m_{28}$$

  To verify the safety of the situation, we need to perform the following calculation:

  $$\mathrm{NR}(pq, \mathcal{E})$$

  Given that the resulting value has reverted to zero, we can conclude that the situation has once again become dangerous. There is a potential for a collision between the trains in sections S1 and S8.

As can be observed, any change in the configuration of the railway station (such as the configuration of the turnout switches or the colour of the semaphores) or the position of the trains requires updating the abstract polynomials p and q. This process can be very labor-intensive and prone to potential errors.

3.2. Implementation with the library

Here, we present the implementation of the interlocking system for the previously discussed railway station, based on the model [14] and utilizing the library introduced in this paper. Compared to the previous section, the implementation of the interlocking system is very straightforward and intuitive. It does not require a mathematical background for either the static or dynamic components. Our goal in this section is to demonstrate the ease of implementing and using an interlocking system with our library, especially in comparison to the previous section. Therefore, we will not delve into the details of each function provided by this library. All these details will be covered in subsequent sections of the manuscript.

Static Aspect. The static part can be easily managed using our library `RailwayStationDesign.cocoa5`. This library includes commands such as `DefineSemaphore`, `DefineTurnout`, and `DefineCrossover` to establish the topology of the railway station. The sequence of commands in CoCoA would be as follows (these commands can be found in the file `example source.txt` at http://u.uma.es/fTY/):

```
source "RailwayStationDesign.cocoa5";
DefineSemaphore(1,2);
DefineSemaphore(4,5);
DefineSemaphore(7,8);
DefineSemaphore(4,3);
DefineSemaphore(10,9);
DefineSemaphore(10,11);
DefineSemaphore(7,6);
DefineSemaphore(12,13);
DefineTurnout(2,3,9);
DefineCrossover(5,11,6,12);
GenerateRailwayStation("example.cocoa5");
```

The first command, `source ''RailwayStationDesign.cocoa5''`, imports our library. It is intuitive to observe how the next commands define the sections where each semaphore, turnout, and crossover are located in Fig. 3. Once the static aspect of the railway station is defined, our library `RailwayStationDesign.cocoa5` includes a command, `GenerateRailwayStation`, which creates a new specialized library to handle the interlocking system for this specific railway station (dynamic aspect). As can be seen, our final command creates a library for the railway station in Fig. 3, which we have named `example.cocoa5` in this example (this file can be downloaded at http://u.uma.es/fTY/).

We would like to emphasize that, as can be observed, there is no need to define the ring $\mathcal{A}$ and the list $\mathcal{E}$. These are computed internally based on the commands. It is also important to note that an understanding of algebra is not necessary to specify any of this.

Dynamic Aspect. Using the previously generated library `example.cocoa5` (Static Aspect), we have various commands to establish the position of the trains on the railway station and the configuration of the semaphores, turnouts, and crossovers. We will consider the different scenarios illustrated in Fig. 4.

- Scenario A. Next, we will configure the scenario A in Fig. 4. This requires us to modify the colour of the light signals `L[2]`, `L[4]`, `L[5]`, `L[7]`, and the position of the switch for the crossover `C[1]`. We can achieve this using the following operations:

  ```
  source "example.cocoa5";
  SetColor(L[2],"R");
  SetColor(L[4],"R");
  SetColor(L[5],"R");
  SetColor(L[7],"R");
  SetCrossoverSwitch(C[1],"D");
  PlaceTrain([1]);
  PlaceTrain([10]);
  IsSafe();
  ```

  The first command, `source ''example.cocoa5''`, imports the library related to the interlocking system for the specific railway station shown in Fig. 3. Once this library is imported, semaphores are set to green by default, and the switches of the turnouts and crossovers are set to the straight track position by default. This configuration can be changed using the commands `SetColor`, `SetCrossoverSwitch`, and `SetTurnout` in the `example.cocoa5` library. Additionally, when the library is imported, it is established by default that there are no trains in the railway station. This library includes the commands `PlaceTrain` and `RemoveTrain` to position trains on the railway station. The final command, `IsSafe()`, checks if the situation is safe.
- Scenario B. The train previously placed on section S1 is now positioned on sections S1 and S2. We can translate these changes using the following commands:

  ```
  RemoveTrain([1]);
  PlaceTrain([1,2]);
  ```

- Scenario C. This involves a new train in section S8. This requires the following commands:

  ```
  PlaceTrain([8]);
  IsSafe();
  ```

- Scenario D. In this scenario, the light signal L6 has been adjusted to display red. To reflect this change, we use the next commands:

  ```
  SetColor(L[6],"R");
  IsSafe();
  ```

- Scenario E. In this scenario, the light signal L7 has been set to green, and the switch of the crossover C1 has been set to the straight track position. To accommodate these changes, we apply the following commands:

```
SetColor(L[7],"G");
SetCrossoverSwitch(C[1],"S");
IsSafe();
```

As demonstrated, our library eliminates the need for the user to manually update the monomials p and q since these are computed internally through the natural operations provided by our library. Consequently, the user managing the interlocking system is not required to comprehend the intricate mathematical details of the model described in [14].

In summary, the generated library `example.cocoa5` allows for very straightforward and intuitive management of the station's control elements and the positioning of trains, without the need to resort to changes in abstract monomials or normal forms. This means that, thanks to this library, a person without knowledge of algebra can manage the interlocking system.

4. Architecture. An overview.

We have developed a library in *CoCoA* that simplifies the implementation and management of interlocking systems, effectively addressing all the drawbacks outlined in the previous section. As demonstrated in the prior example, implementing and managing these systems without the aid of the library, and relying solely on the mathematical framework developed in [14], can lead to certain challenges. However, as we illustrated, our library is designed to effectively solve all of these issues.

An overview of our proposal is illustrated in Fig. 5. We provide two files in *CoCoA* (both files can be downloaded at http://u.uma.es/fTY/):

`RailwayStationDesign.cocoa5`. This is a library in *CoCoA* that allows to define the static aspects of the railway station. User only need to input the semaphores, turnout and crossovers in the railway station (by means of operations in the library, `DefineSemaphore`, `DefineTurnout` and `DefineCrosover`) and the library automatically computes the ring $\mathcal{A}$ and the list $\mathcal{E}$ of polynomials in this ring. This library allows to define the static aspect of the railway station. Besides, the library has an important operation `GenerateRailwayStation` that generates a file, that is a new library that allows the user to change all the dynamic aspect of the railway station: that is, the interlocking system for this specific railway station.

`Template.cocoa5`. This is a file that the library `RailwayStationDesign.cocoa5` uses with the operation `GenerateRailwayStation` to generate the library that contains the interlocking system for the specific railway station.

The operation `GenerateRailwayStation`, located in the `RailwayStationDesign.cocoa5` file, is integral to the implementation of the railway interlocking system. This operation produces a library that embodies the dynamic attributes of the railway station. The library comprises the following functions and operations:

- Commands for defining the colour of light signals and the positions of switches for turnouts and crossovers, namely `SetColor`, `SetTurnoutSwitch`, and `SetCrossoverSwitch`.
- Commands for managing train placement and removal, specifically `PlaceTrain` and `RemoveTrain`.
- Functions for understanding the configuration of the railway station. These include `GetColor`, which provides the colour of a light signal, and `GetTurnoutSwitch` and `GetCrossoverSwitch`, which offer information on switch positions for turnouts or crossovers. Additionally, there is a function, `CanPass`, that determines if a train can move from one section to an adjacent one given the current configuration of the railway station.
- The function `IsSafe` is employed to assess the safety conditions within the railway station.

5. Methodology

In this section, we will expose the steps to implement and use an interlocking system for a specific railway station by means of the library we propose. This involves two main tasks:

- Implementation of the interlocking for the railway station (see Section 5.1). This corresponds to the static aspect discussed in Section 2. This task outputs a library in *CoCoA* for the interlocking system specific to the railway station.
- Use of the interlocking system (see Section 5.2). Once the interlocking system is implemented, we can use it to verify the safety of different scenarios that occur in the railway system. This corresponds to the dynamic aspect discussed in Section 2.

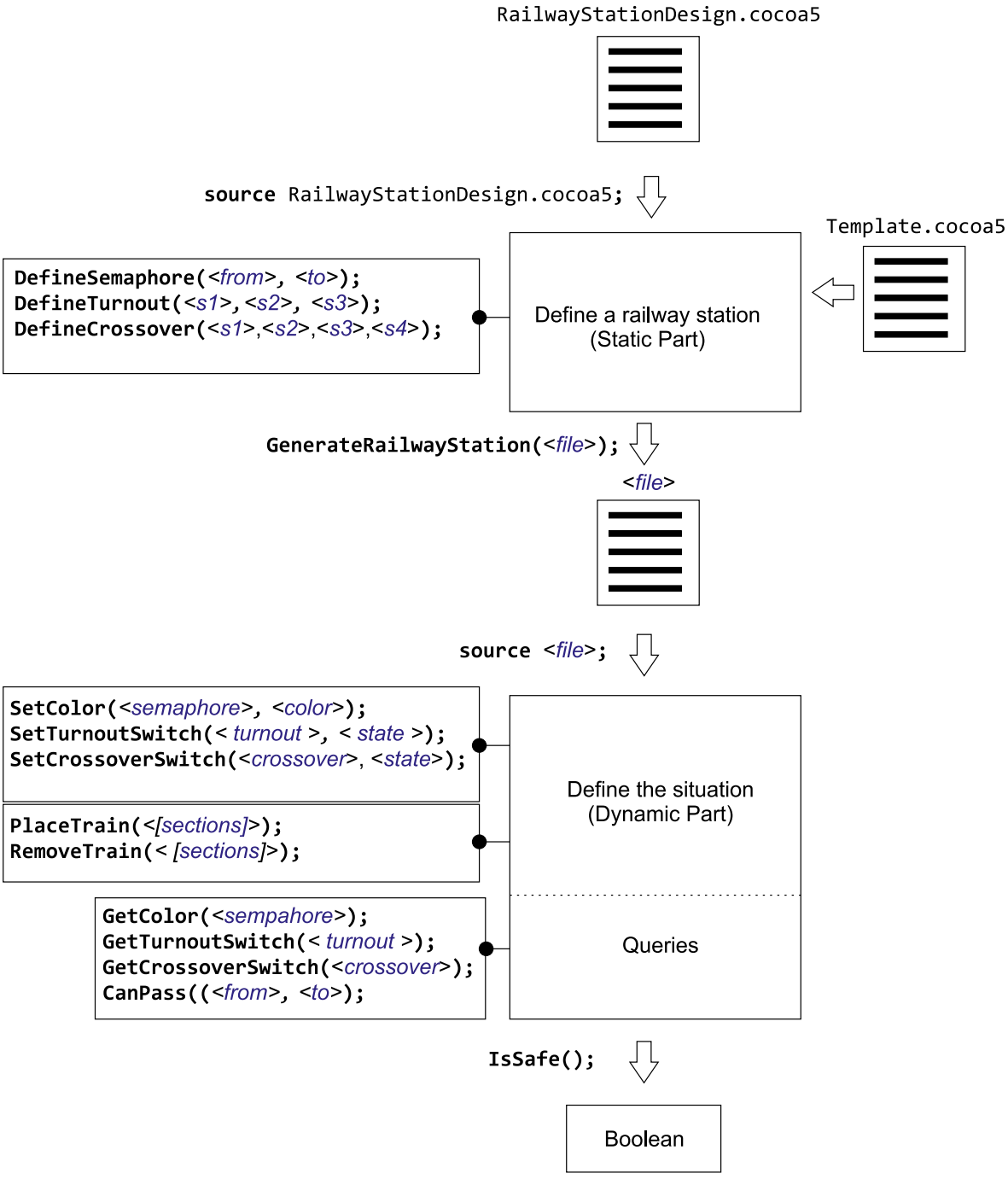

Fig. 5. Architecture.

5.1. Implementation of the interlocking system for a railway station.

In this section, we will define the static structure of the railway station. This involves the following steps:

Step 1. Import the library `RailwayStationDesign.cocoa5` using the command in *CoCoA*:

```
source "RailwayStationDesign.cocoa5";
```

Step 2. Declare the placement of semaphores, turnouts, and crossovers in relation to the sections of the station using the following commands:

- `DefineSemaphore(`⟨*from*⟩`,` ⟨*to*⟩`);`. This command declares that there is a semaphore between sections ⟨*from*⟩ and ⟨*to*⟩. This semaphore is named `L[`n`]`, where n is the number of semaphores declared so far.
- `DefineTurnout(`⟨$s1$⟩`,` ⟨$s2$⟩`,` ⟨$s3$⟩`);`. This command declares that there is a turnout connecting the sections ⟨$s1$⟩, ⟨$s2$⟩ and ⟨$s3$⟩. This turnout is named `D[n]`, where n is the number of turnouts declared so far.
- `DefineCrossover(`⟨$s1$⟩`,` ⟨$s2$⟩`,` ⟨$s3$⟩`,` ⟨$s4$⟩`);`. This command declares that there is a crossover connecting the sections ⟨$s1$⟩, ⟨$s2$⟩, ⟨$s3$⟩ and ⟨$s4$⟩. This crossover is named `C[`n`]`, where n is the number of crossovers declared so far.

Step 3. Generate a specific library that represents the interlocking system for the railway station. This is done using the command:

`GenerateRailwayStation` "(⟨*file*⟩)"; where "⟨*file*⟩" denotes the name of the file in which the interlocking system will be stored.

Following these steps, for the railway station depicted in Fig. 3, we use the following commands:

```
source "RailwayStationDesign.cocoa5";
DefineSemaphore(1,2);
DefineSemaphore(4,5);
DefineSemaphore(7,8);
DefineSemaphore(4,3);
DefineSemaphore(10,9);
DefineSemaphore(10,11);
DefineSemaphore(7,6);
DefineSemaphore(12,13);
DefineTurnout(2,3,9);
DefineCrossover(5,11,6,12);
GenerateRailwayStation("example.cocoa5");
```

5.2. Using the interlocking system

Once we have implemented the interlocking system for the railway station according to the instructions described in the previous section, we can use it to verify whether possible scenarios in this railway station are safe or not. We will proceed with this by following these steps:

Step 1. Firstly, we will import the `example.cocoa5` library into the CoCoA environment:`RailwayStationDesign.cocoa5` using the command in *CoCoA*:

`source` "⟨*file*⟩";

where "⟨*file*⟩" denotes the name of the file in which the interlocking system is stored, generated according to the steps described in the previous section.

Step 2. Once step 1 is completed, all semaphores are set to green by default, and all the switches of the turnouts and crossovers are set to the straight track position by default. We can establish the colour for a specific semaphores, and the status of the switch for a specific turnouts or crossovers, by means of the following commands:

- `SetColor(`⟨*semaphore*⟩`,` ⟨*colour*⟩`);`. This command sets the semaphore *semaphore* to green if *colour* is `''G''`, and red if *colour* is `''R''`.
- `SetTurnoutSwitch(`⟨*turnout*⟩`,` ⟨*position*⟩`);`. This command sets the switch of the turnout ⟨*turnout*⟩ to the straight track position if ⟨*position*⟩ is `''S''`, and to the diverted track position if ⟨*position*⟩ is `''D''`.
- `SetCrossoverSwitch(`⟨*crossover*⟩`,` ⟨*position*⟩`);`. This command declares that there is a crossover connecting the sections ⟨$s1$⟩, ⟨$s2$⟩, ⟨$s3$⟩ and ⟨$s4$⟩.

Step 3. When the library is imported, it is established by default that there are no trains in the railway station. We can place and remove trains by means of the following commands:

- `PlaceTrain([`⟨$s1, s2, \ldots, s_n$⟩`]);`. This command establish a new train on sections $s_1, \ldots, s_n$.
- `RemoveTrain([`⟨$s1, s2, \ldots, s_n$⟩`]);`. This command removes the train placed on $s_1, \ldots, s_n$ from the station.

Step 4. Verify whether the scenario defined in Step 2 and Step 3 is safe using the command:

```
IsSafe();
```

The system will output `true` if the scenario is safe, and `false` if it is not.

Step 5. Repeat Steps 2, 3, and 4 whenever there is a change in the railway station.

Following these steps, for the scenario A depicted in Fig. 3, we use the following commands:

```
source "example.cocoa5";
SetColor(L[2],"R");
SetColor(L[4],"R");
SetColor(L[5],"R");
SetColor(L[7],"R");
SetCrossoverSwitch(C[1],"D");
PlaceTrain([1]);
PlaceTrain([10]);
IsSafe();
```

6. Technical details of the library `RailwayStationDesign.cocoa5`

In this section, we delve into the implementation details of the library `RailwayStationDesign.cocoa5`. Our library provides some functions and operations which modifies internal variables in the library.

6.1. Variables

The library utilizes the following internal variables to characterize the railway station:

`edges`. This refers to a list of edges, which are pairs of sections, representing the set E of the railway station.

In the example in Fig. 3, we have that `edges` is:

```
[[1,2], [2,1], [4,5], [5,4], [7,8], [8,7], [4,3], [3,4], [10,9], [9,10],
[10,11], [11,10], [7,6], [6,7], [12,13], [13,12], [2,3], [2,9], [3,2],
[9,2], [5,6], [5,12], [11,12], [11,6], [6,5], [12,5], [12,11], [6,11]]
```

Each time the operations `DefineSemaphore`, `DefineTurnout`, and `DefineCrossover` are executed, the variable `edge` is updated accordingly.

`L`. This refers to a list of integers that represent the colour light signals within the railway station.

In the example in Fig. 3, we have that `L` is:

```
[1, 3, 5, 7, 9, 11, 13, 15]
```

The value `L[`i`]` indicates the edge that ith-colour light signal controls. If this colour light signal governs the sections from S_a to S_b, then we have that the following:

- `edges[L[`i`]]` is $[a, b]$.
- `edges[L[`i`]+1]` is $[b, a]$.

In the example depicted in Fig. 3, `L[2]` represents the colour light signal that connects section S4 to section S5, and its value is 3. Therefore, we have:

- `edges[3]` is `[4,5]`.
- `edges[4]` is `[5,4]`.

Each time the operation `DefineSemaphore` is executed, the variable `L` is updated accordingly.

`D`. This refers to a list of integers that represent the turnouts within the railway station.

In the example in Fig. 3, we have that `D` is:

```
[17]
```

The value `D[i]` indicates the edges that ith-turnout controls. If this turnout connects sections from S_a to S_b or S_c, if it is possible to pass from section S_a to section S_b, we have that:

- `edges[D[i]]` is $[a, b]$.
- `edges[D[i]+1]` is $[a, c]$.
- `edges[D[i]+2]` is $[b, a]$.
- `edges[D[i]+3]` is $[c, a]$.

For example `D[1]` (the turnout connecting sections S2, S3 and S9) is 17, we have that:

- `edges[17]` is `[2,3]`.
- `edges[18]` is `[2,9]`.
- `edges[19]` is `[3,2]`.
- `edges[20]` is `[9,2]`.

Each time the operation `DefineTurnout` is executed, the variable `D` is updated accordingly.

`C`. This refers to a list of integers that represent the crossover within the railway station.

In the example in Fig. 3, we have that `C` is:

```
[21]
```

The value `C[i]` indicates the edges that ith-crossover controls. If this crossover connects sections from S_{a1}, S_{a2} to S_{b1} or S_{b2}, we have that:

- `edges[C[i]]` is $[a1, b1]$.
- `edges[C[i]+1]` is $[a1, b2]$.
- `edges[C[i]+2]` is $[a2, b2]$.
- `edges[C[i]+3]` is $[a2, b1]$.
- `edges[C[i]+4]` is $[b1, a1]$.
- `edges[C[i]+5]` is $[b2, a1]$.
- `edges[C[i]+6]` is $[b2, a2]$.
- `edges[C[i]+7]` is $[b1, a2]$.

For example `C[1]` is 21, the crossover connecting sections S5 and S11 to S6 and S12. We have that:

- `edges[21]` is `[5,6]`.
- `edges[22]` is `[5,12]`.
- `edges[23]` is `[11,12]`.
- `edges[24]` is `[11,6]`.
- `edges[25]` is `[6,5]`.
- `edges[26]` is `[12,5]`.
- `edges[27]` is `[12,11]`.
- `edges[28]` is `[6,11]`.

Each time the operation `DefineCrossover` is executed, the variable `C` is updated accordingly.

6.2. Commands

`DefineSemaphore(`$\langle from \rangle$**`,`** $\langle to \rangle$**`);`**. This defines a semaphore between sections $\langle from \rangle$ and $\langle to \rangle$. It updates the lists `edges` and `L`.

`DefineTurnout(`$\langle s1 \rangle$ **`,`** $\langle s2 \rangle$**`,`** $\langle s3 \rangle$**`);`**. This defines a turnout between sections $\langle s1 \rangle$, $\langle s2 \rangle$ and $\langle s3 \rangle$. It updates the lists `edges` and `D`.

`DefineCrossover(`$\langle s1 \rangle$ **`,`** $\langle s2 \rangle$ **`,`** $\langle s3 \rangle$ **`,`** $\langle s4 \rangle$**`);`**. This defines a crossover between the sections $\langle s1 \rangle$, $\langle s2 \rangle$, $\langle s3 \rangle$ and $\langle s4 \rangle$.

`GenerateRailwayStation(`$\langle file \rangle$ **`);`**. It generates a file which includes functions in CoCoA to implement an interlocking system for the railway station defined. It includes code related to the definition of the ring $\mathcal{A}$, the list $\mathcal{E}$, the polynomials p and q, and functions related to state the configuration of the railway station (colour of semaphores and switch of the turnouts and crossovers), and to state the position of the trains in the railway station. It uses the file `template.cocoa5` to make this task.

7. Technical details of the library that implements the interlocking system

In this section, we will explore the intricacies of the library's implementation, which facilitates the operation of a railway station's interlocking system and is generated by the `GenerateRailwayStation` operation.

7.1. Variables

The library uses the following internal variables to characterize the situation of a railway station.

`edges, L, D, C.` These variables correspond to those outlined in the previous section of the `RailwayStationDesign.cocoa5` library.

`trains.` This refers to a list of the trains in the railway station. A train is encapsulated as a list of sections. When the library is loaded, this variable is initialized with value `[]` since it is considered that there are no trains in the railway station. Each time the operations `PlaceTrain` and `RemoveTrain` are executed, the variable `trains` is updated accordingly. In the scenario `B` in Fig. 4, we have that `trains` is `[[1,2], [10]]`, since there is a train that occupy sections 1, 2, and there is another train that occupy section 10.

`E.` This refers to the list $\mathcal{E}$. This variable is static and operations in the library do not modify it. In the example in Fig. 3, we have that `E` is (compare it with $\mathcal{E}$ described in the example in Section 3.1):

```
[l[1]*l[2]*t[1]+m[1]*m[2]*t[1]*t[2], l[1]*m[2]*t[1]+m[1]*m[2]*t[1]*t[2],
l[1]*l[2]*t[2]+m[1]*m[2]*t[1]*t[2], l[2]*m[1]*t[2]+m[1]*m[2]*t[1]*t[2],
l[3]*l[4]*t[4]+m[3]*m[4]*t[4]*t[5], l[3]*m[4]*t[4]+m[3]*m[4]*t[4]*t[5],
l[3]*l[4]*t[5]+m[3]*m[4]*t[4]*t[5], l[4]*m[3]*t[5]+m[3]*m[4]*t[4]*t[5],
l[5]*l[6]*t[7]+m[5]*m[6]*t[7]*t[8], l[5]*m[6]*t[7]+m[5]*m[6]*t[7]*t[8],
l[5]*l[6]*t[8]+m[5]*m[6]*t[7]*t[8], l[6]*m[5]*t[8]+m[5]*m[6]*t[7]*t[8],
l[7]*l[8]*t[4]+m[7]*m[8]*t[3]*t[4], l[7]*m[8]*t[4]+m[7]*m[8]*t[3]*t[4],
l[7]*l[8]*t[3]+m[7]*m[8]*t[3]*t[4], l[8]*m[7]*t[3]+m[7]*m[8]*t[3]*t[4],
l[9]*l[10]*t[10]+m[9]*m[10]*t[9]*t[10], l[9]*m[10]*t[10]+m[9]*m[10]*t[9]*t[10],
l[9]*l[10]*t[9]+m[9]*m[10]*t[9]*t[10], l[10]*m[9]*t[9]+m[9]*m[10]*t[9]*t[10],
l[11]*l[12]*t[10]+m[11]*m[12]*t[10]*t[11],
l[11]*m[12]*t[10]+m[11]*m[12]*t[10]*t[11],
l[11]*l[12]*t[11]+m[11]*m[12]*t[10]*t[11],
l[12]*m[11]*t[11]+m[11]*m[12]*t[10]*t[11],
l[13]*l[14]*t[7]+m[13]*m[14]*t[6]*t[7],
l[13]*m[14]*t[7]+m[13]*m[14]*t[6]*t[7],
l[13]*l[14]*t[6]+m[13]*m[14]*t[6]*t[7],
l[14]*m[13]*t[6]+m[13]*m[14]*t[6]*t[7],
l[15]*l[16]*t[12]+m[15]*m[16]*t[12]*t[13],
l[15]*m[16]*t[12]+m[15]*m[16]*t[12]*t[13],
l[15]*l[16]*t[13]+m[15]*m[16]*t[12]*t[13],
l[16]*m[15]*t[13]+m[15]*m[16]*t[12]*t[13],
l[17]*l[19]*t[2]+m[17]*m[19]*t[2]*t[3],
l[17]*m[19]*t[2]+m[17]*m[19]*t[2]*t[3],
l[18]*l[20]*t[2]+m[18]*m[20]*t[2]*t[9],
l[18]*m[20]*t[2]+m[18]*m[20]*t[2]*t[9],
l[17]*l[19]*t[3]+m[17]*m[19]*t[2]*t[3],
l[19]*m[17]*t[3]+m[17]*m[19]*t[2]*t[3],
l[18]*l[20]*t[9]+m[18]*m[20]*t[2]*t[9],
l[20]*m[18]*t[9]+m[18]*m[20]*t[2]*t[9],
l[21]*l[25]*t[5]+m[21]*m[25]*t[5]*t[6],
l[21]*m[25]*t[5]+m[21]*m[25]*t[5]*t[6],
l[22]*l[26]*t[5]+m[22]*m[26]*t[5]*t[12],
l[22]*m[26]*t[5]+m[22]*m[26]*t[5]*t[12],
l[23]*l[27]*t[11]+m[23]*m[27]*t[11]*t[12],
l[23]*m[27]*t[11]+m[23]*m[27]*t[11]*t[12],
 l[24]*l[28]*t[11]+m[24]*m[28]*t[6]*t[11],
 l[24]*m[28]*t[11]+m[24]*m[28]*t[6]*t[11],
 l[21]*l[25]*t[6]+m[21]*m[25]*t[5]*t[6],
 l[25]*m[21]*t[6]+m[21]*m[25]*t[5]*t[6],
```

```
l[22]*l[26]*t[12]+m[22]*m[26]*t[5]*t[12],
l[26]*m[22]*t[12]+m[22]*m[26]*t[5]*t[12],
l[23]*l[27]*t[12]+m[23]*m[27]*t[11]*t[12],
l[27]*m[23]*t[12]+m[23]*m[27]*t[11]*t[12],
l[24]*l[28]*t[6]+m[24]*m[28]*t[6]*t[11],
l[28]*m[24]*t[6]+m[24]*m[28]*t[6]*t[11],
t[1]^2, t[2]^2, t[3]^2, t[4]^2, t[5]^2, t[6]^2, t[7]^2, t[8]^2, t[9]^2,
t[10]^2, t[11]^2, t[12]^2, t[13]^2, t[5]*t[11]]
```

p. This variable refers to the monomial p in the algebraic model and contains information of the configuration of the railway station. When the library is just loaded, all switches of turnouts and crossovers is in straight position, and the colour of semaphores is green. Each time the operations `SetColor`, `SetTurnoutSwitch` and `SetCrossoverSwitch` are executed, the variable `p` is updated accordingly. In the scenario `B` in Fig. 4, we have that `p` is:

```
l[1]*l[2]*l[4]*l[5]*l[6]*l[8]*l[10]*l[11]*l[12]*l[14]*
l[15]*l[16]*l[17]*l[19]*l[22]*l[24]*l[26]*l[28]*m[3]*m[7]*m[9]*m[13]*
m[18]*m[20]*m[21]*m[23]*m[25]*m[27]
```

As previously explained, this monomial includes variables of the type `l[`i`]` and `m[`i`]`. Considering that `edges[`i`]` is `[`a`,`b`]`, then a train can transition from section a to section b if and only if the variable `l[`i`]` is in `p`. Conversely, if a train cannot transition from section a to section b, then the variable `m[`i`]` will be in `p`. For instance, given that `edges[5]` is `[7,8]`, and `l[5]` is in `p`, a train can transition from section 7 to 8. Conversely, considering that `edges[20]` is `[9,2]`, and `l[20]` is not in `p`, a train cannot transition from section 9 to section 2.

q. This refers to the monomial q in the algebraic model. As previously explained, it encapsulates information about the train positions: if a train occupies the section i, then `q` will contain the variable `t[`i`]`. Upon initial loading of the library, when no train is present in the railway station, the value `q` defaults to 1. In the scenario `B` in Fig. 4, we have that `q` is `t[1]*t[2]*t[10]`, since sections 1, 2, 10 are occupied by trains.

E2. This refers to a list of polynomials that encapsulates information of the trains that occupy more than one section. This list is useful for the task of updating p when a train occupy several sections. If a train occupies the edge `edge[`i`]` (that is to say, it occupies the sections defined by `edge[`i`]`), the variable `E2` contains a polynomial `l[`i`]+m[`i`]`. Upon initial loading of the library, when no train is present in the railway station, the value `E2` defaults to `[]`. In the scenario `B` in Fig. 4, we have that `E2` is `[l[1]+m[1],l[2]+m[2]]`, because the edges `edges[1]` (that is `[1,2]`) and the edges `edges[2]` (that is `[2,1]`) are occupied by trains.

7.2. Functions

The library encompasses functions that allow for querying about the configuration of the railway station, as well as operations to establish a specific situation within it.

7.2.1. Functions to ascertain the state of colour light signals, turnouts and crossovers

The colour of the semaphores, as well as the state of the switches of the turnouts and crossovers, can be ascertained by analysing the polynomial `p`.

`GetColor(`⟨*semaphore*⟩**`).`** As stated previously, the light signal ⟨*semaphore*⟩ displays a green light if `l[`⟨*semaphore*⟩`]` is present in the monomial `p`. Conversely, the light signal shows a red light if `m[`⟨*semaphore*⟩`]` appears in the monomial `p`. In *CoCoA*, we can ascertain whether the variable `l[`⟨*semaphore*⟩`]` is present in the monomial `p` (in other words, if the light signal ⟨*semaphore*⟩ displays a green light) using the following statement:

```
IsDivisible(p,l[⟨semaphore⟩])
```

In the railway station depicted in Fig. 3, `L[5]` is assigned the value 9 and `L[8]` is assigned the value 15 (refer to the value of `L` in Section 7.1). In the situation `B` in Fig. 4, the variables `m[9]` and `l[15]` are included in the monomial `p` (see the value of `p` in Section 7.1). As a result, the light signal `L[5]` displays a red light, while the light signal `L[8]` shows a green light, consistent with scenario `B` in Fig. 4.

`GetTurnoutSwitch(`⟨*turnout*⟩**`).`** The turnout switch ⟨*turnout*⟩ is in the straight position if `l[`⟨*turnout*⟩`]` is present in the monomial `p`. If it is not present, the turnout ⟨*turnout*⟩ is in the diverted position, and `m[`⟨*turnout*⟩`]` will appear in the monomial `p`. We can determine whether the variable `l[`⟨*turnout*⟩`]` is present in the monomial `p` (i.e., if the turnout switch ⟨*turnout*⟩ is in the straight position) using the following statement in *CoCoA*,:

`IsDivisible(p,l[`⟨*turnout*⟩`])`

In the railway station depicted in Fig. 3, we have that the turnout `D[1]` is 17 (see the value of `D` in Section 7.1). In scenario `B` in Fig. 4 we have that `l[17]` is in `p` (see the value of `p` in Section 7.1). As a result, the switch of the turnout `D[1]` is in the straight position.

`GetCrossoverSwitch(`⟨*crossover*⟩**`)`**. In the same way, the crossover switch ⟨*crossover*⟩ is in the straight position if `l[`⟨*crossover*⟩`]` is present in the monomial `p`. If it is not present, the turnout ⟨*crossover*⟩ is in the diverted position, and `m[`⟨*crossover*⟩`]` will appear in the monomial `p`. We can determine whether the variable `l[`⟨*crossover*⟩`]` is present in the monomial `p` (i.e., if the crossover switch ⟨*crossover*⟩ is in the straight position) using the following statement in *CoCoA*:

`IsDivisible(p,l[`⟨*crossover*⟩`])`

In scenario `B` in Fig. 4, we have that the crossover `C[1]` is 21 (see the value of `C` in Section 7.1). Besides, we have that `m[17]` is in `p` (see the value of `p` in Section 7.1). As a result, the switch of the crossover `C[1]` is in the diverted position.

`CanPass(` ⟨*from*⟩ **`,`** ⟨*to*⟩ **`)`**. In general, to verify if a train can move from section ⟨*from*⟩ to section ⟨*to*⟩ (where `[` ⟨*from*⟩ `,` ⟨*to*⟩`]` belongs to `edges`), it suffices to check for `l[`i`]`, where i is the index of the element `[` ⟨*from*⟩ `,` ⟨*to*⟩`]` in the list `edges`. We can determine it using the following statement in *CoCoA*:

`IsDivisible(p,l[Index(`⟨*from*⟩`,`⟨*to*⟩`)])`

where `Index` is a function that outputs the index of the element `[`⟨*from*⟩`,`⟨*to*⟩`]` in the list `edges`.

7.3. Commands to establish the state of light signals, turnouts and crossovers

We can establish the colour of the light signals, as well as the status of the switches for the turnouts and crossovers, by adjusting the monomial `p`.

`SetColor(`⟨*semaphore*⟩`,`⟨*colour*⟩**`)`**. This operation requires the inclusion or exclusion of the variables `l[`⟨*semaphore*⟩`]` and `l[`⟨*semaphore*⟩`]` in the monomial `p`, depending on the value of ⟨*colour*⟩. Essentially, it comprises two steps:

- First, it removes the potential variables `l[`⟨*semaphore*⟩`]` and `m[`⟨*semaphore*⟩`]` from the monomial `p`. This is accomplished in *CoCoA* with the following statement:
  `p:=NR(p,[l[`⟨*semaphore*⟩`]-1, m[`⟨*semaphore*⟩`]-1);`
- Subsequently, it incorporates the variable `l[`⟨*semaphore*⟩`]` into `p` if ⟨*colour*⟩ is green; if ⟨*colour*⟩ is red, it incorporates the variable `m[`⟨*semaphore*⟩`]` (see explanation of `L` in Section 6.1). This is accomplished in *CoCoA* with the following statements:
  `if` ⟨*colour*⟩`=''G'' then p:=p*l[`⟨*semaphore*⟩`]; endif;`
  `if` ⟨*colour*⟩`=''R'' then p:=p*m[`⟨*semaphore*⟩`]; endif;`

`SetTurnoutSwitch(`⟨*turnout*⟩`,`⟨*position*⟩**`)`**. This operation requires the inclusion or exclusion of the variables `l[`⟨*turnout*⟩`]`, ... `l[`⟨*turnout*⟩ + 3`]`, and `m[`⟨*turnout*⟩`]`, ... `m[`⟨*turnout*⟩ + 3`]` in the monomial `p`, depending on the value of ⟨*position*⟩ (see explanation of `D` in Section 6.1). Essentially, it comprises two steps:

- First, it removes the potential variables `l[`⟨*turnout*⟩`]`, ... `l[`⟨*turnout*⟩ + 3`]`, and `m[`⟨*turnout*⟩`]`, ... `m[`⟨*turnout*⟩ + 3`]` from the monomial `p`. This is accomplished in *CoCoA* with the following statement:
  `p:=NR(p,[l[`⟨*turnout*⟩`]-1, m[`⟨*turnout*⟩`]-1, l[`⟨*turnout*⟩ + 2`]-1, m[`⟨*turnout*⟩ + 2`]-1, l[`⟨*turnout*⟩ + 3`]-1, m[`⟨*turnout*⟩ + 3`]-1]);`
- Subsequently, it incorporates some of the aforementioned variables depending on the value ⟨*position*⟩, by multiplying them with p.
  - if ⟨*position*⟩ is straight, then it incorporates the variables `l[`⟨*turnout*⟩`]`, `l[`⟨*turnout*⟩ + 2`]`, `m[`⟨*turnout*⟩ + 1`]` and `m[`⟨*turnout*⟩ + 3`]`. This is accomplished in *CoCoA* with the following statement:
    `p:=p*l[`⟨*turnout*⟩`]*l[`⟨*turnout*⟩ + 2`]*m[`⟨*turnout*⟩ + 1`]*m[`⟨*turnout*⟩ + 3`];`
  - if ⟨*position*⟩ is diverted, then:
    `p:=p*m[`⟨*turnout*⟩`]*m[`⟨*turnout*⟩ + 2`]*l[`⟨*turnout*⟩ + 1`]*l[`⟨*turnout*⟩ + 3`];`

SetCrossoverSwitch(⟨*crossover*⟩,⟨*position*⟩). This operation requires the inclusion or exclusion of the variables l[⟨*crossover*⟩], ...l[⟨*crossover*⟩ + 7], and m[⟨*crossover*⟩], ...m[⟨*crossover*⟩ + 7] in the monomial p, depending on the value of ⟨*position*⟩ (see explanation of C in Section 6.1). Essentially, it comprises two steps:

- First, it removes the potential variables l[⟨*crossover*⟩], ...l[⟨*crossover*⟩+7], and m[⟨*crossover*⟩], ...m[⟨*crossover*⟩+7] from the monomial p. This is accomplished in *CoCoA* with the following statement:
  p:=NR(p,[l[⟨*crossover*⟩]-1, m[⟨*crossover*⟩]-1, l[⟨*crossover*⟩ + 2]-1, m[⟨*crossover*⟩ + 2]-1, l[⟨*crossover*⟩ + 3]-1, m[⟨*crossover*⟩ + 3]-1, l[⟨*crossover*⟩ + 4]-1, m[⟨*crossover*⟩ + 4]-1, l[⟨*crossover*⟩ + 5]-1, m[⟨*crossover*⟩ + 5]-1, l[⟨*crossover*⟩ + 6]-1, m[⟨*crossover*⟩ + 6]-1, l[⟨*crossover*⟩ + 7]-1, m[⟨*crossover*⟩ + 7]-1]);
- Subsequently, it incorporates some of the aforementioned variables depending on the value ⟨*position*⟩, by multiplying them with p.
  - if ⟨*position*⟩ is straight, then:
    p:=p*l[⟨*crossover*⟩]*l[⟨*crossover*⟩ + 4]* l[⟨*crossover*⟩ + 2]*l[⟨*crossover*⟩ + 6]* m[⟨*crossover*⟩ + 1]*m[⟨*crossover*⟩ + 5]* m[⟨*crossover*⟩ + 3]*m[⟨*crossover*⟩ + 7];
  - if ⟨*position*⟩ is diverted, then:
    p:=p*m[⟨*crossover*⟩]*m[⟨*crossover*⟩ + 4]* m[⟨*crossover*⟩ + 2]*m[⟨*crossover*⟩ + 6]* l[⟨*crossover*⟩ + 1]*l[⟨*crossover*⟩ + 5]* l[⟨*crossover*⟩ + 3]*l[⟨*crossover*⟩ + 7];

7.4. Commands to define the position of trains

These operations update the variables trains, q and E2 to maintain the meaning we provided in Section 6.1.

PlaceTrain([⟨$s_1, s_2, \dots, s_n$⟩]). This operation comprises the following steps:

- Append the list [s_1, s_2, ... s_n] to trains.
- Update the variable q to incorporate the sections that the new train occupies. This is done as follows:
  q:=q*t[s_1]*t[s_2]*... *t[s_n];
- Append the polynomials l[s_1]-m[s_1],...,l[s_n]-m[s_n] to the list E2.

RemoveTrain([⟨*sections*⟩]). This operation comprises the following steps:

- Remove the list [s_1, s_2, ... s_n] from trains.
- Update the variable q to remove the sections that the train occupied. This is done as follows:
  q:=q/(t[s_1]*t[s_2]*... *t[s_n]);
- Remove the polynomials l[s_1]-m[s_1],...,l[s_n]-m[s_n] from the list E2.

7.5. Commands to determine if the situation is dangerous

We can determine whether a situation is safe or dangerous using the boolean function IsSafe()

According to [14], we first need to transform the situation into an equivalent one where trains occupy only one section. This is achieved by updating p with the value NR(p,E2). We then check if NR(p*q,E)=0. In summary, this function is with the following statement: return NR(NR(p,E2)*q,E)<> 0;

8. Conclusions

In [14], we introduced a linear and fast algorithm for implementing railway interlocking systems. This algorithm is based on the representation of the railway station through polynomials. With this representation, determining whether a situation in a railway station is dangerous becomes equivalent to calculating the residue of the division of a monomial over a set of polynomials.

Despite the speed and linearity of this mathematical framework, which gives it a significant advantage over others, it has several practical drawbacks: implementing an interlocking system for a railway station based on this algebraic method [14] requires a strong background in algebra, polynomial division, and Gröbner bases to implement and manage the interlocking system; the model requires handling a vast number of unintuitive polynomials in many variables and the use of the interlocking system can be extremely labor-intensive and prone to errors, even for expert mathematicians.

To address this, we have presented a library in *CoCoA* in this paper. This library simplifies the implementation of interlocking systems based on our mathematical framework. As a result, even users without any mathematical knowledge can implement and manage a railway interlocking system in an easy and straightforward manner. We believe that this type of library is essential for the practical use of this model. As future work, it can be expanded to develop a graphical environment that enables the implementation of the interlocking system for a specific railway station through a visual design of this railway station. All files related to the library, the illustrative example in Section 3, and the files from a real Spanish railway station (Algodor Railway Station) to highlight the practical advantages of our library with real-world scenarios can be downloaded at http://u.uma.es/fTY/.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgements

We thank the anonymous reviewers for their useful suggestions and corrections which have improved the quality of the paper.

Data availability

No data was used for the research described in the article.

References

[1] N. Bešinović, Resilience in railway transport systems: a literature review and research agenda, Transp. Rev. 40 (2020) 457–478, http://dx.doi.org/10.1080/01441647.2020.1728419.
[2] Y. Zhou, J. Wang, H. Yang, Resilience of transportation systems: concepts and comprehensive review, IEEE Trans. Intell. Transp. Syst. 20 (2019) 4262–4276, http://dx.doi.org/10.1109/TITS.2018.2883766.
[3] F. Bădău, Railway interlockings – a review of the current state of railway safety technology in europe, Saf. Secur. Traffic Rev. 34 (3) (2022) http://dx.doi.org/10.7307/ptt.v34i3.3992.
[4] T. Brandejský, V. Fábera, M. Leso, Integral railway interlocking system and its assessment according to European standards, Acta Polytech. CTU Proc. 43 (2023) 1–5, http://dx.doi.org/10.14311/APP.2023.43.0001.
[5] L. Huang, The past, present and future of railway interlocking system, in: 2020 IEEE 5th International Conference on Intelligent Transportation Engineering (ICITE), 170-174. http://dx.doi.org/10.1109/ICITE50838.2020.9231438.
[6] R. Cavada, et al., Analysis of relay interlocking systems via SMT-based model checking of switched multi-domain kirchhoff networks, in: 2018 Formal Methods in Computer Aided Design (FMCAD), 1-8. http://dx.doi.org/10.23919/FMCAD.2018.8603013.
[7] H. Lian, X. Wang, A. Sharma, M.A. Shah, Application and study of artificial intelligence in railway signal interlocking fault, Informatica 46 (2022) 343–354, http://dx.doi.org/10.31449/inf.v46i3.3961.
[8] A. Fantechi, G. Gori, A.E. Haxthausen, C. Limbrée, Compositional verification of railway interlockings: comparison of two methods, in: Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, 2022, pp. 3–19, http://dx.doi.org/10.1007/978-3-031-05814-1_1.
[9] A.E. Haxthausen, A. Fantechi, Compositional verification of railway interlocking systems, Form. Asp. Comput. 35 (1) (2023) 4, http://dx.doi.org/10.1145/3549736.
[10] I. Ustoglu, Topology based automatic formal model generation for point automation systems, Control. Syst. Eng. (2024) https://www.academia.edu/Documents/in/Railway_Interlocking_System.
[11] G. Lukács, T. Bartha, Conception of a formal model-based methodology to support railway engineers in the specification and verification of interlocking systems, in: 2022 IEEE 16th International Symposium on Applied Computational Intelligence and Informatics, SACI, 2022, pp. 000283–000288, http://dx.doi.org/10.1109/SACI55618.2022.9919532.
[12] A. Hernando, R. Maestre, E. Roanes-Lozano, A new algebraic approach to decision making in a railway interlocking system based on preprocess, Math. Probl. Eng. 2018 (2018) 4982974, http://dx.doi.org/10.1155/2018/4982974.
[13] A. Hernando, J.L. Galán-García, G. Aguilera-Venegas, A novel way to build expert systems with infinite-valued attributes, AIMS Math. 9 (2) (2024) 2938–2963, http://dx.doi.org/10.3934/math.2024145.
[14] A. Hernando, E. Roanes-Lozano, J.L. Galán-García, G. Aguilera-Venegas, Decision making in railway interlocking systems based on calculating the remainder of dividing a polynomial by a set of polynomials, Electron. Res. Arch. 31 (10) (2023) 6160–6196, http://dx.doi.org/10.3934/era.2023313.
[15] A. Hernando, J.L. Galán-García, G. Aguilera-Venegas, A fast and general algebraic approach to railway interlocking system, Aims Math. 9 (3) (2024) 7673–7710, http://dx.doi.org/10.3934/math.2024373.
[16] A. Hernando, J.L. Galán-García, G. Aguilera-Venegas, S. Nazary, An interlocking system determining the configuration of rail traffic control elements to ensure safety, AIMS Math. 9 (8) 21471–21495. http://dx.doi.org/10.3934/math.20241043.
[17] J. Abbott, A.M. Bigatti, CoCoALib: a C++ library for doing Computations in Commutative Algebra, 2019, Available from: http://cocoa.dima.unige.it/cocoalib.