Verification of Interlockings: from Control Tables to Ladder Logic Diagrams

Wan Fokkink and Paul Hollingshead
University of Wales Swansea
Department of Computer Science
Singleton Park, Swansea SA2 8PP, Wales
emails: w.j.fokkink@swan.ac.uk, prh@sucs.swan.ac.uk

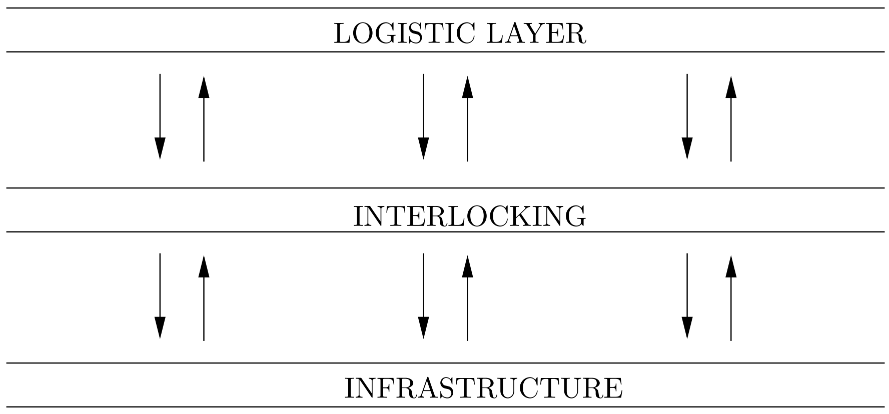
Abstract

Dependency relations between objects in a railway yard are tabulated in control tables. An interlocking, which guarantees validity of these dependencies, can be implemented in ladder logic. We transform a ladder logic diagram into a Boolean formula, so that validity of the dependencies in the control tables can be verified using a theorem prover. Time copies and invariants are added to the formula, to relate it more firmly to its ladder logic diagram. Program slicing is applied to reduce the size of the formula.

1 Introduction

Railway signalling has evolved over the last 150 years to provide an extremely high level of safety, both for the railway staff and their customers. In the last twenty years, traditional approaches to safety have been modified by the introduction of microprocessors into a number of areas of signalling. New standards call for ever-improved methods for developing safety systems and software. In addition, safety cases have to be presented for all new equipment and installations. These trends, together with the increasing commercial pressures, result in a demand on those in the signalling supply industries to produce ever more innovative systems, software, and methodologies.

A railway yard is built from objects such as points, signals, track circuits, and level crossings. The control and management of such a railway yard consists of three separate layers. First, there is the infrastructure, where the objects can be in different states; points can be in normal or reverse position, a signal can show several aspects, track circuits can be occupied or not, level crossings can be open or closed. Second, in the logistic layer, human experts devise control instructions for the railway yard, in order to guide the movements of trains. Third, it has to be guaranteed that the execution of these control instructions does not jeopardise safety; that is, collisions and derailments have to be avoided. This is done by means of an interlocking, which is a medium between the infrastructure at the one side, and the logistic layer and its interfaces at the other side. The interplay of the three layers of a railway yard, logistic layer, interlocking, and infrastructure, is depicted below.



An interlocking is the embodiment of a number safety principles, according to which a train moves through a railway yard. If a control instruction violates such a safety principle, then the interlocking automatically discards or delays it. In the early days of railways, interlockings were mechanical devices, which for instance linked the positions of two separate points. This century saw the implementation of interlockings by means of electronic relays. In more recent years, the implementation of interlockings has shifted to software. In general, programs for interlockings are still under the influence of methods in electrical engineering; that is, they mimic the working of electrical relays in minute detail. Interlockings must take into account timing aspects such as delays of electrical devices.

The safety principles and basic rules that underly an interlocking are specified by means of control tables [7]. In such control tables, the dependencies between the separate objects in a railway yard are expressed explicitly. For example, if a signal shows red, then this imposes restrictions on the range of aspects of other signals, in order to warn train drivers that they may encounter a red signal. Or if a track circuit is occupied, then the positions of points in this track circuit are fixed. The interlocking of a railway yard is meant to implement the restrictions that are listed in the control tables of this railway yard. Each railway station is supplied with its own interlocking, based on the particularities of the road map of the railway tracks.

The states of objects in a railway yard can be captured by means of Boolean variables. For example, given a signal s, we introduce a variable x_a for each possible aspect a of s; x_a is true if and only if signal s shows aspect a. Similarly we can capture the position of points, whether or not a track circuit is occupied, and whether or not a level crossing is closed. An interlocking for a railway yard essentially consists of a large number of assignments of the form $v = \phi$ with v a variable and ϕ a Boolean formula, where each such equation expresses a dependency between the objects in the railway yard. Security in a railway yard is ensured by the execution of subsequent control cycles, using the assignments of the interlocking. Each control cycle starts with reading new values for the input variables, which are determined by inputs from the logistic layer and the infrastructure. Next, the values of internal and output variables are determined by means of the assignments of the interlocking. The output values are transmitted to the outside world, where they are used in managing the states of the objects in the railway yard. Finally, the remaining time of the control cycle is used to carry out a number of housekeeping tasks, including measures to assure integrity of the process. In the initial control cycle, all values of variables are taken to be false, which is guaranteed to yield a safe situation.

Ladder logic is a graphical notation used by signalling engineers from Westinghouse Signals to define sets of Boolean equations, which implement interlockings. It is descended from the design notation which has been used to define electrical circuits made up of relays. At a growing number of international railway stations, computer equipment based on ladder logic diagrams is used in order to ensure safe movement of trains. The production of a ladder logic diagram for a specific railway yard from its control tables requires meticulous human labour, and even for a small railway station the resulting ladder logic diagram is large and complicated. A bottleneck in the strive for a hundred percent security of train movements is the possible existence of flaws in the ladder logic diagrams. Westinghouse Signals follow a thorough checking and testing procedure for their interlockings. After the initial production of a ladder logic diagram from a set of control tables, the diagram is proofread by an independent party. The creator of the diagram is informed of the parts of his diagram where flaws were discovered, but not of the nature of these mistakes. The flaws are repaired, after which the diagram is passed on to the proofreader again, et cetera. This procedure is repeated until no mistakes are spotted anymore. A simulation environment has been implemented, which is able to animate the behaviour of the ladder logic diagram, via a graphical display. Before the ladder logic diagram is installed, first it is tested using the simulation environment, so that flaws may be detected without jeopardising safety.

In this article we explore an alternative method, from Groote, Koorn, and van Vlijmen [4], to verify whether a ladder logic diagram satisfies all the restrictions that are imposed by the control tables of a railway yard. A ladder logic diagram can be turned into a large Boolean formula Φ (see Hollingshead [5]), which basically consists of the conjunction of its assignments. Furthermore, each dependency relation in the control tables can be captured by a Boolean formula ρ. For each such safety requirement, we want to be sure that it holds in all circumstances. In order to ensure that the interlocking is a correct implementation of the dependencies in the control tables, it needs to be verified for each dependency relation ρ that the formula $\Phi \wedge \neg\rho$ is not satisfiable. Since the satisfiability problem of Boolean formulae is NP-complete (see e.g., [8]), no polynomial-time algorithm is known to solve this question in full generality. However, recently developed theorem provers have shown that in many cases the question of (un)satisfiability of large Boolean formulae need not be problematic. The program described in [5] transforms a ladder logic diagram into a Boolean formula that can serve as input to the theorem prover HeerHugo [3].

Certain requirements involve time, for example, "if a track circuit was occupied less than ten seconds ago, then ...". For such requirements, we want to describe the dependencies between the objects in the railway yard that are imposed by an interlocking for a period of time. If a requirement ρ involves n control cycles, then we make time copies $\Phi_0, \Phi_1, ..., \Phi_n$ of the Boolean formula Φ, where Φ_i describes the dependencies in the railway yard i control cycle ago. The conjunction $\Phi_0 \wedge \cdots \wedge \Phi_n$ describes the dependencies between the objects in the railway yard in the last n control cycles. We desire that the formula $\Phi_0 \wedge \cdots \wedge \Phi_n \wedge \neg\rho$ is invalid for all possible values of variables in this expression.

In general, a ladder logic diagram consists of a long list of assignments. Given a requirement ρ, a program slicing technique (see e.g,, [12]) can be applied to reduce the size of the ladder logic diagram. First, it is determined which assignments may contribute to the value of ρ, by means of a recursive procedure. Next, the remaining assignments are deleted from the ladder logic diagram. The result is usually a considerably smaller set of assignments, which can be transformed into a Boolean formula Φ as before. Finally, satisfiability of $\Phi \wedge \neg\rho$ can be checked using a theorem prover.

For the verification of requirements in a control table, it is not always sufficient to restrict to a limited number of time copies. Certain properties of objects in a railway yard cannot be derived from the assignments in the ladder logic diagram as such. They can only be derived from these assignments if it is taken into account that initially the values of all variables in the ladder logic diagram are false. Such a property is called an invariant of the ladder logic diagram, and can be added to the list of assignments.

Ladder logic diagrams are closely related to the methodology of Vital Processor Interlocking (VPI). Groote, Koorn, and van Vlijmen [4] presented a specification of VPI, together with a manual verification of desirable properties of VPI. Furthermore, they introduced the verification method described above for VPIs. Fokkink [2] formulated classes of safety criteria for VPIs, and verified instantiations of these classes for the railway yard at Hoorn-Kersenboogerd, by means of the Stålmarck [9, 10] and HeerHugo [3] theorem provers. We note that such classes of safety criteria can be obtained from the control tables of a railway yard. Mertens [6] repeated the verification exercise for the larger railway yard at Heerhugowaard, and needed invariants to express that points are never both in normal and in reverse position. Stålmarck and Säflund [11] applied the Stålmarck theorem prover to verify properties of an interlocking system used by the Swedish state railways.

2 Control Tables

A railway yard consists of a collection of linked railway tracks, supplied with objects such as signals, points, and level crossings. Each of the objects in a railway yard can attain a certain number of states:

- a railway track is either occupied or unoccupied;
- a signal shows, for example, red, yellow, or green;
- points are in reverse or in normal position;
- a level crossing is open or closed.

Figure 1 depicts a schematic view of part of the road map of a railway yard. Note that each separate object in the railway yard is provided with a unique identifier.

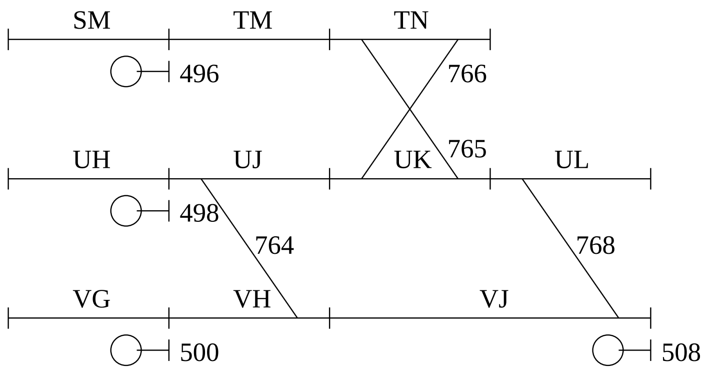


Figure 1: Layout of a railway yard

In order to avoid dangerous situations in the railway yard, certain combinations of states are to be avoided. For example, if a level crossing is open, then the track where it is situated must be unoccupied, in order to avoid a collision of a train with passing street traffic. Such dependencies are expressed in control tables. Nowadays, the manual preparation of control tables for a specific railway yard requires considerable expertise. We list some of the main control methods.

- Route locking: A route consists of a collection of adjacent track circuits, together with their points and signals. A route can be claimed, to guarantee safe passage of a train along this route. First, it is verified that the route does not conflict with existing routes. Second, the points in the route are locked in the required positions. Third, it is verified that the track circuits in the route are all unoccupied. Then the entrance signal is cleared, meaning that it is allowed to show yellow or green.

- Approach locking: If a route is locked, the entrance signal of the route is cleared, and the track circuit immediately approaching this signal is occupied, then the approach locking mechanism is applied. Approach locking prevents immediate cancellation of the route.

- Flank protection: Where possible, the flanks of a route that has been locked should be protected. That is, around the route, signals should show red, and points should be in such positions that they do not give immediate access to the route, even if it is expected that no train will pass such a signal or such points in the near future.

ROUTE No.	EXIT	ASPECT	SIGNAL AHEAD	TRACKS	POINTS NORMAL	POINTS REVERSE
508 A	500	Y	500 AT R	VJ VH VG	764 768	
		G	500 AT Y			
508 B-1	498	Y	498 AT R	VJ VH UJ UH (UK if 765N or 766R)	768	764
		G	498 AT Y			
508 B-2	498	Y	498 AT R	VJ UL UK UJ UH	764 765 766	768
		G	498 AT Y			
508 C	496	Y	496 AT R	VJ UL UK TN TM SM	766	765 768
		G	496 AT Y			

Figure 2: A control table

Figure 2 presents a somewhat simplified example of a control table for the railway yard in Figure 1, which deals with several aspects route locking. It distinguishes the four possible routes that await behind signal 508.

1. Route 508A consists of the track circuits VJ, VH, and VG, and requires that the points 764 and 768 are in normal position. The exit signal of this route is 500, and if signal 508 shows green, then signal 500 should show yellow or green.

2. Route 508B-1 consists of the track circuits VJ, VH, UJ, and UH, and requires that the points 764 and 768 are in reverse and in normal position, respectively.

Furthermore, if points 765 are in normal position, or if points 766 are in reverse position, then track circuit UK is also included in the route, The exit signal of this route is 498, and if signal 508 shows green, then signal 498 should show yellow or green.

3. Route 508B-2 consists of the track circuits VJ, UL, UK, UJ, and UH, and requires that the points 764, 765, and 766 are in normal position, and that the points 768 are in reverse position. The exit signal of this route is 498, and if signal 508 shows green, then signal 498 should show yellow or green.

4. Route 508C consists of the track circuits VJ, UL, UK, TN, TM, and SM, and requires that the points 765 and 768 are in reverse position, and that points 766 are in normal position. The exit signal of this route is 496, and if signal 508 shows green, then signal 496 should show yellow or green.

For exhaustive examples, and more background information on control tables, the reader is referred to [7, Chapter 3].

3 Ladder Logic Diagrams

3.1 Interlockings and Boolean Formulae

As explained in the introduction, an interlocking is a safety layer between the logistic layer and the infrastructure of a railway yard. It is meant to encapsulate instructions that are considered to be unsafe. An interlocking implements a number of safety principles and basic dependencies between the objects in the railway yard.

In general, an interlocking operates in discrete control cycles. In one control cycle, it reads the values for its input variables from the logistic layer and the infrastructure. These values together with the values of variables in the previous control cycle are used to compute the values of variables in the current control cycle. Finally, it transmits the values of its output variables to the logistic layer and the infrastructure, and enters the next control cycle. Basically, an interlocking consists of a list of equations of the form $x_i := e_i$, where the x_i are distinct variables, and e_i is an expression, containing variables, that is used to compute the value of x_i in each control cycle.

Intuitively, the variables that are used in an interlocking represent the possible states of the separate objects in a railway yard. For instance, for each signal s and each aspect a there is a variable x_a that expresses whether signal s shows aspect a; x_a is true if and only if s shows aspect a. Thus, the variables in an interlocking all have the Boolean values 'true' and 'false'. Boolean logic, which is an ideal formalism for the implementation of interlockings, is constructed from:

- the constants 1 ('true') and 0 ('false');

- a finite set of variables;

- the binary operators conjunction $\wedge$ (‘and’) and disjunction $\vee$ (‘or’);
- the unary operator negation $\neg$ (‘not’);
- the implication $\rightarrow$ and the bi-implication $\leftrightarrow$.

It is well-known that each formula in Boolean logic can be transformed into an equivalent formula in disjunctive normal form (see e.g., [8]), being a formula

$$\bigvee_{i=1}^{n} \Phi_i$$

where the Φ_i are of the form $\ell_1 \wedge \cdots \wedge \ell_m$ with each ℓ_j either a variable x or a negation of a variable $\neg x$.

3.2 Ladder Logic

Ladder logic is graphical notation used by signalling engineers to define sets of Boolean equations, which together implement an interlocking. It consists of expressions R that are defined inductively as follows. We assume a set of variables, with typical element x.

$\overset{x}{\text{--[ ]--}}$ represents the variable x.

$\overset{x}{\text{--[/]--}}$ represents the negation $\neg x$.

$R\ \overset{x}{\text{--[ ]--}}$ represents the conjunction $R \wedge x$.

$R\ \overset{x}{\text{--[/]--}}$ represents the conjunction $R \wedge \neg x$.

R_1 -- + -- represents the disjunction $R_1 \vee R_2$.
R_2 --┘

Since each formula is equivalent to a formula in disjunctive normal form, it is easy to see that each formula can be described by means of an expression R.

A ladder logic diagram consists of a vertical list of assignments of the form

$$\begin{array}{ccc} R_1 & : & x_1 \\ R_2 & : & x_2 \\ & \vdots & \\ R_n & : & x_n \end{array}$$

meaning that the variable x_i, called a coil, is assigned the truth value of R_i for $i = 1, ..., n$. Each expression $R_i : x_i$ is called a rung.

A ladder logic diagram incorporates five types of variables:

- Input variable: Its value is determined by the environment (i.e., the logistic layer and the infrastructure).
- Output variable: Its value is computed by means of the ladder logic diagram, and passed on to the environment.
- Latch: Its value is computed by means of the ladder logic diagram, and not passed on to the environment, but only used in the computation of values of other variables.
- Timer: This variable is either on or off, which is determined by the value of its trigger. If it is off, then its value is 0. If it is on, then its value is increased by one with every control cycle, until it reaches a preset duration, after which its trigger is switched off (i.e., is assigned the value 0).
- Trigger: Indicates whether a certain timer operation is on or off.

Input variables, output variables, latches and triggers have Boolean values (i.e., 0 or 1), while the values of timers are natural numbers. Each output variable, latch or trigger x is the coil of exactly one assignment $R : x$ in the ladder logic diagram. Input variables and timers are not allowed as coils. Only input variables, latches and triggers are allowed to occur in the left-hand sides of rungs (so output variables and timers are excluded from these left-hand sides).

A ladder logic diagram implements an interlocking, and is executed in discrete control cycles as follows. In the initial control cycle, the values of all variables are set to 0. Now suppose that we have computed the values of the variables in the n-th control cycle. Then the values of the variables in the $(n + 1)$-th control cycle are computed as follows;

1. The inputs from the logistic layer and the infrastructure are read, to determine the values of the input variables.
2. The values of timers that are off are set to 0, and the values of timers that are on are increased by one. If a timer reaches its duration, its trigger is set to 0.
3. The assignments in the ladder logic diagram are applied, from top to bottom, to compute the values of the output, latch, and trigger variables.

When the computation of the values of all variables in control cycle $n+1$ is finished, the values of output variables are transmitted to the logistic layer and the infrastructure. After completion of control cycle $n + 1$, the same procedure is repeated to determine the values of the variables in control cycle $n + 2$, et cetera.

It is important to note that occurrences of a coil x_i in the first i equations of the ladder logic diagram refer to the value of x_i in the n-th control cycle, while occurrences of x_i in the remaining equations of the ladder logic diagram refer to the value of x_i in the $(n + 1)$-th control cycle.

4 Verification of Safety Requirements

4.1 Construction of a Boolean Formula

Hollingshead [5] implemented a tool that transforms a ladder logic diagram into a Boolean formula. Assume a ladder logic diagram that consists of a list of rungs $R_i : x_i$ for $i = 1, ..., n$. Each expression R_i is transformed into a Boolean formula Φ_i, by induction on its structure. Some care has to be taken whether an occurrence of a coil x_i in an R_j refers to the value of this coil in the current or in the previous control cycle. If $i \leq j$ then x_i refers to the previous control cycle, and if $i > j$, then x_i refers to the current control cycle in which it was assigned the truth value of R_i. Therefore, occurrences of coil x_i in expressions R_j for $j \leq i$ and for $j > i$ are represented in the formulae Φ_j by distinct variable names: $x_i(1)$ and $x_i(0)$, respectively. Furthermore, input variables y in the Φ_j are represented as $y(0)$, to denote that they refer to the current control cycle. Finally, the ladder logic diagram is transformed into the Boolean formula

$$\bigwedge_{i=1}^{n} (\Phi_i \leftrightarrow x_i(0)).$$

Dependency relations in control tables can also be transformed into Boolean formulae ρ, by means of a (manual) conversion. For each such formula ρ we want to know that it is guaranteed by the interlocking at all times. For this purpose we need to verify that the formula

$$(\bigwedge_{i=1}^{n} (\Phi_i \leftrightarrow x_i(0))) \wedge \neg\rho$$

is not satisfiable; that is, for all possible valuations of the variables in this formula, the value of this formula should be 0.

4.2 Time Copies

A ladder logic diagram L_0 that implements an interlocking contains "loose ends". First, there are the input variables, of which the values are determined by the logistic layer and the infrastructure. The interlocking cannot make any assumptions about the values of these variables; it must guarantee safety at the railway yard for all possible valuations of input variables. Second, there are the values of coils x_i in the previous control cycle, represented in the form $x_i(1)$. It is possible to say something more about the values of these variables; they were determined by the truth values of the expressions R_i in the previous control cycle. In the Boolean formula Φ_0 that is obtained from the ladder logic diagram L_0, this information is not taken into account. This imperfection can be remedied by the construction of a time copy L_1 of L_0, which is obtained by replacing occurrences of variables $y(0)$ and $y(1)$ in L_0 by $y(1)$ and $y(2)$, respectively. Hereby, $y(j)$ refers to the value of the variable y j control cycles ago. The dependencies in the interlocking are represented more accurately by combination of the ladder logic diagrams L_0 and L_1, which are transformed into a formula $\Phi_0 \wedge \Phi_1$.

The combination of L_0 and L_1 contains new loose ends; the coils $x_i(2)$. In order to limit the range of values of these coils, we can construct a time copy L_2, which is obtained by replacing occurrences of variables $y(0)$ and $y(1)$ in L_0 by $y(2)$ and $y(3)$, respectively. The dependencies in the interlocking are represented more accurately by the combination of the ladder logic diagrams L_0, L_1, and L_2, which are transformed into a formula $\Phi_0 \wedge \Phi_1 \wedge \Phi_2$.

The combination of L_0, L_1, and L_2 contains as loose ends the coils $x_i(3)$, et cetera. In general, a time copy L_j for $j > 0$ is obtained from L_0 by replacing occurrences of variables $y(0)$ and $y(1)$ in L_0 by $y(j)$ and $y(j+1)$, respectively. The dependencies in the interlocking can be represented by the combination of ladder logic diagrams L_j for $j = 0, ..., m$, which give rise to a formula

$$\Phi_0 \wedge \Phi_1 \wedge \cdots \wedge \Phi_m.$$

Requirements in control tables can incorporate time delays. The formulae that are obtained from such restrictions contain variables $y(j)$ with $j > 0$. Consider such a formula ρ, and let m be the greatest number such that ρ contains an occurrence of a variable $y(m)$. In order to verify more accurately whether ρ does not hold under any circumstances, we can construct the time copies $\Phi_1, ..., \Phi_m$ of Φ_0, as explained above, and verify that the formula

$$\Phi_0 \wedge \Phi_1 \wedge \cdots \wedge \Phi_m \wedge \neg\rho$$

is not satisfiable.

4.3 Invariants

For the verification of the requirements in control tables, it is not always sufficient to restrict to a limited number of time copies. We explained that the operation of an interlocking based on a ladder logic diagram operates in discrete control cycles. In the initial control cycle, the values of all variables are set to 0, while the values of variables in the $(n+1)$-th control cycle are computed using the values of variables in the n-th control cycle. Thus, the fact that initially the values of variables were set to 0 may influence the values of variables in the future.

In order to verify that the validity of requirements in the control tables is guaranteed by a ladder logic diagram, it is necessary to derive "invariants". Let Φ_0 be the Boolean formula that is obtained from the ladder logic diagram, with time copies $\Phi_1, \Phi_2, \ldots$. An invariant consists of a of Boolean formula I that satisfies the following properties.

1. I holds in the initial control cycle, when the values of all variables are 0.

2. $I' \wedge \Phi_0 \wedge \cdots \wedge \Phi_m \rightarrow I$ is a tautology for some $m \geq 0$, whereby I' is obtained from I by replacing each variable $y(j)$ in I by $y(j+1)$. In other words, if I holds in a certain control cycle, then it can be proved using the equations in the ladder logic diagram that I holds in the following control cycle.

Then it follows by induction that I is always true. In order to verify all requirements in the control tables, it is necessary to prove invariants $I_1, ..., I_k$ (see [6]). Furthermore, we adapt the formula Φ_0 to the form

$$\Phi_0 \wedge I_1 \wedge \cdots \wedge I_k,$$

and we adapt its time copies accordingly.

4.4 Timers

The values of timer are natural numbers, so they are not allowed to occur in ladder logic diagrams. Their meaning is specified outside the ladder logic diagram, where each timer is given a certain duration. If the trigger of a timer has the value 1, then the value of the timer is increased by one with every control cycle, until it reaches its duration, when both the timer and its trigger are assigned the value 0. If the trigger has the value 0, then its timer has the value 0 too. Thus, each timer has an implicit influence on the value of its trigger, and vice versa. This relation between a timer and its trigger can be expressed explicitly by the introduction of a sequence of rungs in the ladder logic diagram. Let *trigger* be the trigger of a timer with duration n, and let the rung for this trigger in the ladder logic diagram be ϕ : *trigger*. Then we introduce fresh Boolean variables y_i for $i = 1, ..., n$, and we replace the rung ϕ : *trigger* in the ladder logic diagram by a string of rungs:

$$\begin{array}{rcl} \phi \wedge \neg y_1 & : & \mathit{trigger} \\ y_2 \wedge \mathit{trigger} & : & y_1 \\ & \vdots & \\ y_n \wedge \mathit{trigger} & : & y_{n-1} \\ \mathit{trigger} & : & y_n. \end{array}$$

These rungs together capture the behaviour of the timer with duration n. Initially the values of the coils *trigger* and y_i for $i = 1, ..., n$ are all 0. If in some control cycle the value of the formula ϕ becomes 1, then *trigger* is assigned the value 1 (because $y_1 = 0$). Hence, in the same control cycle the coil y_n is assigned the value 1. Furthermore, if in some control cycle *trigger* and y_{i+1} have the value 1, then in the next control cycle the coil y_i is assigned the value 1. Then there are two possibilities.

1. The value of the formula ϕ remains 1 in the next $n-1$ control cycles.

   Then *trigger* has the value 1 in the next $n-1$ control cycles, so after i control cycles y_{n-1} is assigned the value 1, for $i = 1, ..., n$. Hence, after $n-1$ control cycles the coil y_1 is assigned the value 1. Then in the n-th control cycle, *trigger* is assigned the value 0. So in the same control cycle, the values of the coils y_i for $i = 1, ..., n$ become 0.

2. The value of the formula ϕ becomes 0 within n control cycles.

   Then the value of *trigger* becomes 0, and so in the same control cycle the values of the coils y_i for $i = 1, ..., n$ become 0.

Hence, the rungs expressed above exactly describe the interplay of the timer and its trigger.

4.5 Program Slicing

In general, a ladder logic diagram that implements an interlocking, including invariants (Section 4.3), timers (Section 4.4), and time copies (Section 4.2), is very large. In order to reduce this ladder logic diagram, we can apply a slicing technique.

Suppose that we want to verify a requirement ρ. Then we are only interested in the values of variables in ρ. So rungs in the ladder logic diagram of which the coils do not influence the values of variables in ρ can be discarded. Such coils can be determined as follows.

1. First, we determine the collection C_0 of variables in ρ that are coils. For each $x \in C_0$, we collect the variables in the rung of x that are coils, and add them to C_0. The resulting collection of coils is denoted by C_1.

2. Suppose that we have constructed the collections C_i of coils for $i \leq k$, where $k > 0$. For each $x \in C_k \backslash C_{k-1}$, we collect the variables in the rung of x that are coils, and add them to C_k. The resulting collection of coils is denoted by C_{k+1}.

This procedure is repeated until $C_{k+1} = C_k$. Then C_{k+1} contains all coils that may influence the values of variables in the requirement ρ. The original ladder logic diagram can therefore be limited to rungs of which the coil is in C_{k+1}. In general the result of this slicing operation is a considerably smaller ladder logic diagram; see [2]. This reduced ladder logic diagram is transformed into a Boolean formula Ψ, as explained in Section 4.1. Then it remains to prove that $\Psi \wedge \neg\rho$ is not satisfiable.

5 Conclusion

An interlocking is a buffer between the logistic layer and the infrastructure of a railway yard, which filters out unsafe instructions. At Westinghouse Signals, interlockings are implemented in ladder logic, on the basis of the control tables of a railway yard. The intensive testing procedure for interlockings is time and money consuming, and although the procedure is thorough and carried out by experts and semi-automated simulation, it does not give a 100% guarantee that an interlocking satisfies the dependencies in its control tables.

We explained how a ladder logic diagram can be converted to a Boolean formula Φ. Furthermore, the dependencies in control tables can be converted to Boolean formulae ρ. A theorem prover can be applied to verify that the formulae $\Phi\wedge\neg\rho$ are not satisfiable. Thus, the cost of verifying interlockings can be reduced, and it can be guaranteed that an interlocking agrees with its control tables.

For a given requirement, we can apply program slicing to reduce the size of the ladder logic diagram, so that it only contains assignments that are related to the requirement. It is important to determine invariants, which are satisfied in all control cycles, owing to the fact that initially the values of all Boolean variables are taken to be 0.

Acknowledgements. Special thanks go to Harry Ryland from Westinghouse Signals for his support. Thanks also go to Bob Ellard and Nick Smith from Westinghouse Signals for technical discussions, and to Jan Friso Groote from the CWI for his permission to use the theorem prover HeerHugo.

References

[1] J.A. Bergstra, W.J. Fokkink, W.M.T. Mennen, and S.F.M. van Vlijmen. *Railway Logic via EURIS*. Quaestiones Infinitae XXII, Zeno Institute of Philosophy, 1997. In Dutch.

[2] W.J. Fokkink. Safety criteria for the vital processor interlocking at Hoorn-Kersenboogerd. In J. Allan, C.A. Brebbia, R.J. Hill, G. Sciutto, and S. Sone, eds., *Proceedings of the 5th Conference on Computers in Railways (COMPRAIL'96), Volume I: Railway Systems and Management*, Berlin, pp. 101–110. Computational Mechanics Publications, 1996.

[3] J.F. Groote. Hiding propositional constants in BDDs. Logic Group Preprint Series 120, Utrecht University, 1994. Available by ftp from `ftp.phil.ruu.nl` as `logic/PREPRINTS/preprint120.ps.Z`.

[4] J.F. Groote, J.W.C. Koorn, and S.F.M. van Vlijmen. The safety guaranteeing system at station Hoorn-Kersenboogerd. In *Proceedings of the 10th IEEE Conference on Computer Assurance (COMPASS'95)*, Gaithersburg, pp. 131–150. IEEE, 1995.

[5] P.R. Hollingshead. Correctness of the implementation of railway control tables by ladder logic. Master's Thesis, University of Wales Swansea, 1998.

[6] J. Mertens. Verifying the safety guaranteeing system at railway station Heerhugowaard. Master's Thesis, University of Utrecht, 1996.

[7] O.S. Nock, ed. *Railway Signalling: A Treatise on the Recent Practise of British Railways.* A&C Black, 1980.

[8] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[9] G. Stålmarck. A note on the computational complexity of the pure classical implication calculus, *Information Processing Letters*, 31:277–278, 1989.

[10] G. Stålmarck. Normalization theorems for full first order classical natural deduction, *Journal of Symbolic Logic*, 56:129–149, 1991.

[11] G. Stålmarck and M. Säflund. Modelling and verifying systems and software in propositional logic. In *Proceedings of the 9th Conference on Computer Safety, Reliability and Security (SAFECOMP'90)*, pp. 31–36. Pergamon Press, 1990.

[12] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3:121–189, 1995.