Formal Methods for Validation and Test Point Prioritization in Railway Signaling Logic

Shiladitya Ghosh, Arindam Das, Nirvik Basak, Pallab Dasgupta, *Senior Member, IEEE*, and Alok Katiyar

***Abstract*—The EN50128 Railway Safety Standard recommends the use of formal methods for proving the correctness of the yard-specific logic, which was developed for electronic signaling and interlocking systems. We present a tool flow, which consists of three components. The core component uses a novel method for automatically generating the relevant safety properties for a yard from its control table. The second component proves the validity of the properties on the application logic by using a new theory of invariant checking. The third component leverages the suite of formal properties to prioritize site acceptance test points. Experimental results are presented on real application data for the yards in India that are demonstrating the performance of the proposed methods.**

***Index Terms*—Railway safety, formal verification.**

I. Introduction

RAILWAY signaling by its inherent safety critical nature has been in the attention of researchers in formal verification for some time [1]. In spite of various prescriptions from the research community, the methods used today for developing the signaling logic for railway yards continue to rely on legacy practices. For many of the large and old railways, such dependence is unavoidable given the large volume of existing assets that use relays and configurations of any given type. A modern railway electronic interlocking (EI) and signaling system is developed on the following foundations:

1) The international railway signaling principles which specifies the rules governing the movement of trains in a railway yard and in block sections. These are broadly the same across the world.
2) Yard-specific data, alternatively called application data, that describes the different routes through which the trains can move in a yard. The application data is compiled into an application logic which is interpreted by the EI equipment.
3) An *executive software* that resides in the EI. The executive software interprets the application logic and controls the signals and point positions in the yard in accordance with the signaling principles.

Manuscript received October 16, 2015; revised February 19, 2016; accepted June 19, 2016. Date of publication August 2, 2016; date of current version February 24, 2017. This work was supported in part by the Indian Railways through the Center for Railways Research, Indian Institute of Technology Kharagpur. The Associate Editor for this paper was D. Work.

S. Ghosh, A. Das, N. Basak, and P. Dasgupta are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur 721 302, India (e-mail: shiladitya.ghosh@cse.iitkgp.ernet.in).

A. Katiyar is with Research Design and Standards Organization, Indian Railways, Lucknow 226 011, India.

Digital Object Identifier 10.1109/TITS.2016.2586512

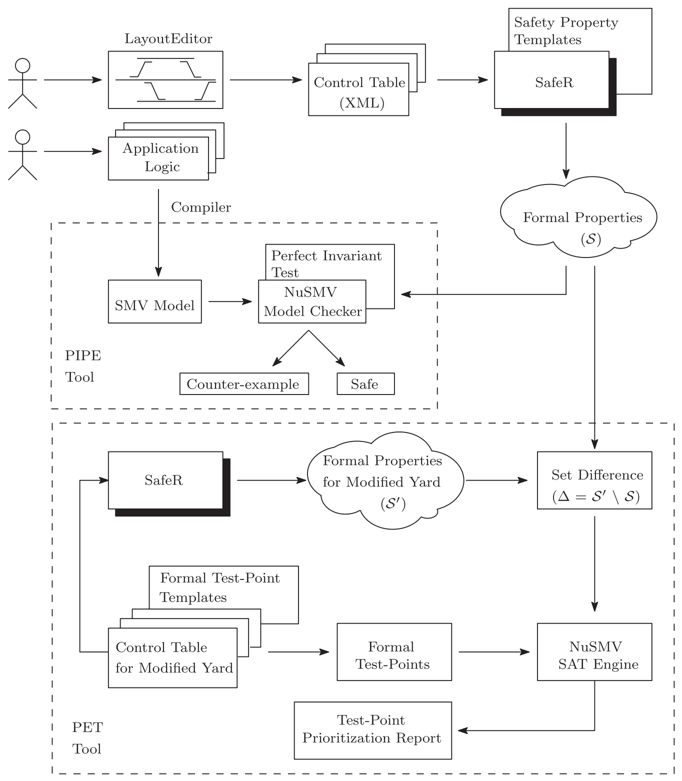
The *executive software* is proprietary in nature and is typically unreadable for the end user, namely the railway company. The application data is prepared by the railway company because they design the yard and decide the choice of routes governing the movement of trains. The compilation of the application data into the application logic is often done by a third party. This distributed nature of development of EI systems typically gets in the way of setting up a formal validation flow for the railway company.

It is important to understand that though the executive software of the EI vendor is certified by independent certification authorities, the configured EI equipment may have bugs in the application logic. There are instances of such bugs in the not so distant past. The Milton Keynes incident (Dec 29, 2008) [2] and the Cootamundra incident (Nov 12, 2009) [3] are among the notable ones.

These incidents highlight the need for formally proving the correctness of the application logic, and this is vindicated by the recommendations in the EN50128 [4] signaling safety standards for using formal methods for this purpose. In the recent past, manufacturers of EI equipment have considered a wide variety of formal approaches, some of which can be found in [5]–[7]. Model checking techniques have also been attempted, for example [8], [9]. An approach based on theorem proving for verification of signaling data can be found in [10]. There are several important differences between these and our work, both in purpose and our approach. This will be explained shortly.

Railway yards are often upgraded, invariably with a corresponding change in the application logic. When a new yard is commissioned, the EI equipment undergoes rigorous on-field testing, but when it is upgraded, such comprehensive testing is typically infeasible for an active yard without major disruptions in the movement of trains. The railway company can leverage its formal verification setup to prove the safety of the modified application logic thereby improving its safety assurance level. Moreover, as we shall show in this paper, formal methods may be used to prioritize the on-field test points in the context of a given upgrade.

The application logic is typically developed through a two step process. In the first step, signal engineers develop a *locking table* (also called a *control table*) from the layout of the yard. In the second step, the application logic is developed from the locking table. The logic developed in this second step is tailored to the combination of relays used in the signaling system, and is *not* standardized. This means that yards in different parts of the country may use different types of relays, and consequently the logic driving these relays also varies from one part of the country to the other.



LayoutEditor
Control Table (XML)
Safety Property Templates
SafeR
Application Logic
Compiler
Formal Properties ($\mathcal{S}$)
Perfect Invariant Test
SMV Model
NuSMV Model Checker
PIPE Tool
Counter-example
Safe
SafeR
Formal Properties for Modified Yard ($\mathcal{S}'$)
Set Difference ($\Delta = \mathcal{S}' \setminus \mathcal{S}$)
Formal Test-Point Templates
Control Table for Modified Yard
Formal Test-Points
NuSMV SAT Engine
PET Tool
Test-Point Prioritization Report

Fig. 1. Developed tool flow.

Therefore, any practical solution for formal validation of the signaling logic developed by a third party must have the capability for generating the formal specification in terms of the combination of relays and signals used in that specific yard. This requirement separates our work from many of the existing literature on formal verification of signaling systems which assume that the logic is given in terms of the core signaling aspects [11], [12].

Fig. 1 shows the tool flow presented in this paper. At the heart of our tool flow is SafeR, an engine that automatically generates formal safety properties for a railway yard as entailed by the signaling principles. These properties are generated in terms of the combinations of relays specific to that yard. As indicated in Fig. 1, our tool flow has two application components that use these properties, namely:

1) The *Perfect Invariant Prover Engine* (PIPE): This tool uses the theory of perfect invariants for proving the generated properties on the application logic. The notion of perfect invariants was introduced in one of our earlier papers [13], but the theory and its application in railway signaling is reported here for the first time.
2) The *Prioritization Engine for Test-points* (PET): After each upgrade in the signaling logic of a yard, on-site tests need to be carried out to make sure that the physical assets (relays, signals, etc) are properly interfaced with the EI logic. The PET tool uses formal methods to prioritize the tests that are more relevant to the modified yard.

At the front end of our tool flow, we have a layout editor that allows the user to enter the layout of the yard using standard terminologies. We also have a compiler that generates a control table for the yard using the layout and the set of specified routes. The control table generated is then fed into SafeR. We also have the option of reading a given control table directly from a given XML input format.

We present experimental results on two railway yards in India to demonstrate the efficacy of each of these components. The actual names of the stations have been deliberately obfuscated.

The paper is organized as follows. Section II introduces the notion of *Strong Safety* and *Perfect Invariants*. Section III presents our approach of automatically generating formal properties following safety principles of railway signaling and interlocking. This is followed by Section IV where the implementation details for the PIPE tool is given. Our approach for test point prioritization and details regarding the PET tool is presented in Section V. Experimental results are presented in Section VI.

II. Strong Safety and Perfect Invariants

Formal properties are broadly of two types, namely *safety properties* and *liveness properties*. Intuitively, safety properties check that bad things never happen and liveness properties check whether good things eventually happen. Safety does not guarantee that the system makes progress–for example, a state of the signaling system where all signals are red is safe, but nothing moves as a consequence. Liveness properties help in establishing that the system eventually progresses, and this is something that every testing procedure must cover. On the other hand, proving a safety property on a system is the equivalent of searching for a run of the system that violates the property. We define the notion of *strong safety* as follows.

Definition 2.1 [Strong Safety]: A system, $\mathcal{M}$, is strongly safe with respect to a safety property, φ, iff all runs of $\mathcal{M}$ satisfy φ regardless of the state at which $\mathcal{M}$ is initialized. □

In other words, a strongly safe system, $\mathcal{M}$, with respect to property φ, will continue to satisfy φ even if it is reset to any arbitrary state. Safety critical systems typically satisfy an inductive version of strong safety (and this is true for railway signaling properties), as explained below. Intuitively, a safety critical system must guarantee that it never transits from a safe state to an unsafe state. Therefore, if φ is a safety property, then the property ψ which states—"*If φ holds at a state, then it also holds in all next states*"—is a strong safety requirement. If we guarantee that the system is always reset to a safe state (that is, one satisfying φ), then the strong safety requirement establishes that the system always stays in a safe state.

We use the theory of perfect invariants to prove strong safety requirements. The notion of perfect invariants was defined in our previous work [13] in a different domain, but the theory was not elaborated. We provide a detailed version of the theory in this paper.

A. Perfect Invariants

In verification our primary concern is to determine whether some execution of the system leads to a bad state. We can define what is a *bad state* by some Boolean combination of state variables. For example, the requirement that signals $S1$ and $S2$

should never be cleared at the same time can be expressed using the Boolean formula:

$$\varphi : \qquad S1 \vee S2.$$

It may be noted that in signaling parlance, a signal S is high (logic 1) when it is RED. Turning a signal from RED to YELLOW or GREEN is called *clearing* the signal. A property such as φ is called an *invariant* of the signaling logic, if φ holds true at all states that are reachable from the initial states of the system. For example, in a signaling network, a safe initial state is one at which all the signals are ON (red).

Invariants may be Boolean properties or *temporal properties*. Boolean properties are defined over a single state, where as temporal properties are defined over a sequence of states. For example, a property that states that a signal $S3$ cannot be cleared if any of the track segments $T1$, $T2$, and $T3$ are occupied can be expressed using the temporal formula:

$$\psi : \qquad (T1 \vee T2 \vee T3) \Rightarrow X(\neg S3).$$

The X operator represents *next time*. The property says that if $T1$ or $T2$ or $T3$ is occupied, then *in the next instance of time* we cannot have $S3$ cleared. The X operator is used to sequence the events and can be used with the implication operator to define cause-effect relationships across time.

In order to prove that a property is an invariant in a state machine defined in logic (such as the signaling logic), we must prove that it holds on *all reachable states* of the state machine. The set of reachable states of a system such as a signaling system can be very large and therefore attempts to examine all these states individually lead to *state explosion problems*, rendering such attempts infeasible for all practical purposes.

Formally, a state transition system, $\mathcal{M}$, is defined as $\mathcal{M} = \langle Q, Q_0, T, \mathcal{AP}, \mathcal{L} \rangle$ where,

- Q is the finite set of states
- $Q_0 \subseteq Q$ is the set of initial states
- $T \subseteq Q \times Q$ is the set of valid transitions
- $\mathcal{AP}$ is the set of atomic propositions
- $\mathcal{L}$: $Q \rightarrow 2^{\mathcal{AP}}$ is the labeling function

Safety properties may be expressed using Linear Temporal Logic (LTL) [14]. The syntax for an LTL formula over a set of atomic propositions can be defined recursively as the following:

$$\varphi := \text{true} | a | \varphi_1 \wedge \varphi_2 | \neg\varphi | X\varphi | \varphi_1 U \varphi_2$$

where $a \in \mathcal{AP}$ and φ_1, φ_2 are LTL formulae. Here X and U denote the *next* and the *until* operator respectively. $G\varphi$ and $F\varphi$ are used to denote the *Always* and *Future* operators. Using the basic temporal operators $F\varphi$ can be written as (true $U\varphi$) and $G\varphi$ can be written as $\neg$(true $U\neg\varphi$).

The semantics of LTL is as follows. We say that an LTL property holds on $\mathcal{M}$ iff φ holds on all paths of $\mathcal{M}$ starting from its initial state. A path of $\mathcal{M}$ is a sequence of states $\pi = p_0, p_1, \ldots$, starting from a start state of $\mathcal{M}$ following the transition relation given as T. We use the notation $\pi \models \varphi$ to denote that the property φ holds on a path π. We use the notation π_k to denote a path starting from p_k. The formal semantics for the basic temporal operators are given as follows:

- $\pi \models X\varphi$ iff $\pi_1 \models \varphi$
- $\pi \models \varphi_1 U \varphi_2$ iff $\exists j$ such that $\pi_j \models \varphi_2$ and $\forall\, i$, $0 \leq i \leq j$ we have $\pi_i \models \varphi_1$

The task of model checking (formal verification) is to determine if any infinite run of M reaches a violation of the property φ. In case a violation is found, the model checker reports the run as a counter-example trace. The sequential depth of φ is the maximum number of time steps over which a match or fail of φ may occur.

In a Bounded Model Checking (BMC) [15] approach, propositional clauses are generated from the transition relation and the negation of the property to determine whether a counter-example exists within a given sequential bound k. Obviously k is greater than the sequential depth of the property φ. To check whether φ fails in any of the initial states, we check the satisfiability of the following formula:

$$I(s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_\tau, s_{\tau+1}) \wedge \neg\varphi(s_1). \qquad (1)$$

Here $I(S)$ is used to represent the initial states of M, and τ is the sequential depth of φ, and $\varphi(s_i)$ denotes that φ is true at state s_i. If Formula 1 is unsatisfiable, then we know that φ is true at all initial states of $\mathcal{M}$.

To prove that φ holds in those states of M that are reachable in k or fewer steps, we check if the following formula is unsatisfiable.

$$I(s_1) \wedge \neg\varphi(s_1) \wedge T(s_1, s_2) \wedge \neg\varphi(s_2)$$
$$\wedge \cdots \wedge T(s_{\tau+k-1}, s_{\tau+k}) \wedge \neg\varphi(s_k).$$

The major disadvantage of this approach is that BMC must keep on increasing the bound k if a counterexample is not found and this can continue until k reaches the diameter[1] of the state transition graph of $\mathcal{M}$, which is in general very large. To get around this problem, inductive approaches for proving invariants have been developed [16].

For example, to prove that ψ is an invariant inductively, we prove that ψ is true in all initial states and we prove that the following formula is unsatisfiable at all states of the system:

$$R(s_i) \wedge \psi(s_i) \wedge T(s_i, s_{i+1}) \wedge \neg\psi(s_{i+1})$$

where $R(s_i)$ denotes that s_i is reachable from the initial states of the system. If this formula is unsatisfiable, then it means every *reachable* state satisfying ψ can only have successors satisfying ψ. If ψ also holds on all initial states, then we may conclude inductively that all reachable states satisfy ψ—that is, ψ is an invariant.

The main challenge in using the above approach is in identifying the set of reachable states, or in other words, determining the truth of $R(s_i)$ for a given s_i. In order to determine whether s_i is reachable from an initial state, we may have to navigate a significant portion of the state space, or equivalently we may

[1]The diameter of a state transition graph is the maximum among the shortest distance between all pairs of states.

have to increase the depth bound of BMC significantly, often leading to scalability problems.

A *perfect invariant* is a property that is true at all states of the system. Let us consider the following property:

$$\eta : \qquad (\psi(s_i) \wedge T(s_i, s_{i+1})) \Rightarrow \psi(s_{i+1}).$$

It is interesting to note that if η is a perfect invariant, and ψ is true at all initial states, then ψ is an invariant. The reverse is not necessarily true and ψ may be an invariant even though η is not a perfect invariant. Nevertheless, if we prove η to be a perfect invariant then we have a less expensive way to prove that ψ is an invariant.

Safety critical systems are traditionally designed in a way that they are fail safe, that is, once the system is in a safe state, then it stays in safe states. This is precisely what the property, η, models for a given safety property, ψ. In other words, for safety critical properties (such as ψ), the corresponding inductive formula (such as η) is typically a perfect invariant, and we exploit this fact to prove signaling properties efficiently.

Example 2.1: Consider the property $\varphi = S1 \vee S2$ defined earlier, which states that signals $S1$ and $S2$ should not be cleared together. The property is not a perfect invariant. This is because the signaling system has states satisfying $(\neg S1) \wedge (\neg S2)$, and whether or not such states are reachable from an initial state is the verification question. Now consider the following property:

$$\varphi' : \qquad (S1 \vee S2) \Rightarrow X(S1 \vee S2).$$

This property, φ', is the inductive version of φ. It may be seen, that unlike φ, this property, φ' should be a perfect invariant. This is because φ' is vacuously true at states where the antecedent is false, and we expect the signaling system to disallow transitions from a safe state to an unsafe state. □

One of the main aspects which separates our work from some of the previous studies on formal validation of signaling logic is that we generate the formal properties automatically in the form of perfect invariants. Automatic generation is made possible by integrating deep knowledge about railway signaling principles, which is the source of the safety requirements, and yard specific information, such as the control table and the combination of relays used in the system. Moreover, the understanding of perfect invariants is built into the tool to make it possible to generate the properties in the inductive form prescribed in this section.

III. Automatic Generation of Strong Safety Properties

The basis of our method for automatic generation of strong safety properties is the railway signaling principles as described in [17]. These principles are to be interpreted on the basis of the type of signaling system employed. Given that the majority of the yards in India use *automatic fixed block* signaling systems, our tool focuses on this type of system. In fixed block signaling systems, a yard is divided into sections and safety of train movement is guaranteed by letting only a single train occupy a section of the yard at a time.

TABLE I
Operational Semantics of the Relays

Relay	***State***	***Meaning***
TPR	De-energized	Proves that a track is occupied
LR	Energized	Proves that a route is initiated
NCR	Energized	Proves that a point is moved to its normal position
RCR	Energized	Proves that a point is moved to its reverse position
WLR	De-energized	Proves that a point is locked
UCR	Energized	Proves that all the points in a route, its overlap and isolation are locked and detected. Also proves that no conflicting routes are set
ASR	De-energized	Proves that the route is locked after all the checks are complete
HR	Energized	Proves that the route is locked and the signal is showing a yellow aspect
LCYR	De-energized	Proves that level crossings are closed and are locked

Traditional fixed block signaling systems are implemented using relays. The combination of relays used in *route relay interlocking* (RRI) and their purpose vary from one zone of the railways to the other. In modern EI systems some of these relays are not physically present anymore, but the application logic is still developed using these relays for legacy reasons. Therefore for verifying application logic developed by a third party, the tool must understand the semantics of the relays and interpret the safety properties on the relay based logic accordingly.

The application logic is written in ladder logic over Boolean variables representing the state of the relays. The properties that our tool generates are expressed in *Linear Temporal Logic* (LTL) [14]. The current prototype uses the set of relays given in Table I. This table shows the state of energy in the relays when the logic variable corresponding to that relay is high (logic 1).

To begin with, let us look into a simple example explaining how a signaling rule can be formally expressed as a LTL formula and why such properties can be treated as perfect invariants.

Example 3.1: Consider the yard, called Yard_1, as shown in Fig. 2. The control table for Yard_1 is also shown in Fig. 3.[2] Let us consider a fundamental safety principle, namely to ensure that no two *conflicting routes*[3] are asserted at the same time. The relay that indicates initiation of route assertion is LR. The relay level interpretation of the safety requirement is therefore to ensure that LR relays of conflicting routes are not energized at the same time. Our tool finds the pairs of conflicting routes automatically from the control table, and interprets the safety properties automatically on the LR relays of all such conflicting routes.

For example, in the yard of Fig. 3, routes $1A$ and $8A$ are conflicting. Corresponding to this pair, the following properties are generated automatically:

$$\psi_1 : (\neg 1A\_\mathrm{LR} \vee \neg 8A\_\mathrm{LR})$$
$$\psi_2 : G\left((\neg 1A\_\mathrm{LR} \vee \neg 8A\_\mathrm{LR}) \Rightarrow X(\neg 1A\_\mathrm{LR} \vee \neg 8A\_\mathrm{LR})\right).$$

[2]A glossary of terms shown in the control table has been provided in the Appendix.

[3]Two routes are conflicting if they share a common section of the railway yard.

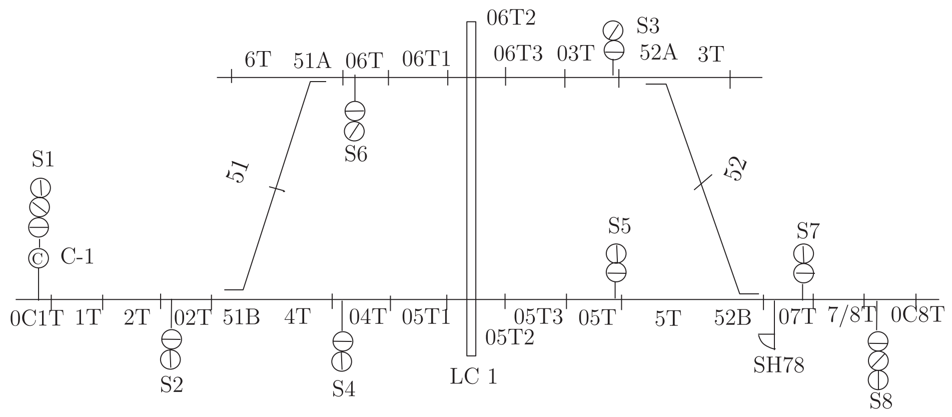


Fig. 2. Schematic view of yard_1.

SL NO	ENTRY SIGNAL	EXIT SIGNAL	ROUTE	ROUTE		OVERLAP				CONTROLLED BY TRACK CIRCUIT	SIGNAL REPLACED BY TRACK CIRCUIT	BACK LOCKED UNTILL TRACK CIRCUIT CLEAR	LEVEL CROSSING	CRANK HANDLES	CONFLICTING ROUTES
				POINT NORMAL	POINT REVERSE	TRACKS	POINT NORMAL	POINT REVERSE	OVERLAP SET						
1	S1	S5	1A	51	---	5T, 07T	52	---	OV-5	1T, 2T, 02T, 4T, 04T, 05T1, 05T2, 05T3, 05T	1T	1T, 2T, 02T, 4T	LC 1	CH1, CH2	C-1A, 4, 8A, 78A
2	S1	S3	1BD	---	51	3T	52	---	OV1-3	1T, 2T, 02T, 4T, 6T, 06T, 06T1, 06T2, 06T3, 03T	1T	1T, 2T , 02T, 4T, 6T	LC 1	CH1, CH2	C-1B, 6, 78A
			1BM			3T, 5T, 07T	---	52	OV2-3				LC 1		C-1B, 6, 8B, 78A, 78B, 8A, 5
3	S3	S7	3	---	52	---	---	---	---	3T, 5T, 07T	3T	3T, 5T		CH2	8B, 78B, 6, C-1B
4	S5	S7	5	52	---	---	---	---	---	5T, 07T	5T	5T		CH2	8A, 78A, 4, C-1A
5	C-1	S5	C-1A	51	---	---	---	---	---	1T	1T	1T, 2T, 02T, 4T	LC 1	CH1	1A, 4, 8A, 78A
6	C-1	S3	C-1B	---	51	---	---	---	---	1T	1T	1T, 2T, 02T, 4T, 6T	LC 1	CH1	1B, 6, 8B, 78B
7	S8	S4	8A	52	---	4T, 02T	51	---	OV-4	7/8T, 07T, 5T, 05T, 05T3, 05T2, 05T1, 04T	7/8T	7/8T, 07T, 5T	LC 1	CH1, CH2	C-1A, 78A, 5, 1A
8	S8	S6	8BD	---	52	6T	51	---	OV1-6	7/8T, 07T, 5T, 3T, 03T, 06T3, 06T2, 06T1, 06T	7/8T	7/8T, 07T, 5T, 3T	LC 1	CH1, CH2	C-1B, 78B, 3
			8BM			6T, 4T, 02T	---	51	OV2-6				LC 1		C-1B, 78B, 3, 1B, C-1A, 1A, 4
9	S4	S2	4	51	---	---	---	---	---	4T, 02T	4T	4T		CH1	1A, C-1A, 5, 78A
10	S6	S2	6	---	51	---	---	---	---	6T, 4T, 02T	6T	6T, 4T		CH1	1B, C-1B, 3, 78B
11	SH78	S4	78A	52	---	---	---	---	---	5T	5T	5T	LC 1	CH2	1A, C-1A, 8A, 5
12	SH78	S6	78B	---	52	---	---	---	---	5T, 3T	5T	5T, 3T	LC 1	CH2	1B, C-1B, 8B, 3

Fig. 3. Control table for yard_1.

The first property, ψ_1, states that the initial state of the system is safe. The second property, ψ_2, is a strong safety property (that is, an inductively posed perfect invariant) which guarantees that the system continues to remain in a safe state when it is in a safe state. A key observation from our study is that all safety requirements in the type of signaling systems considered can be posed and proven as perfect invariants.

The signaling principles for Indian railways [17] consists of rules for setting, canceling and releasing routes. In order to explain the property generation methodology, we focus on the rules for route setting and locking.

1) *Prove that the interlocking is free. To initiate the process of setting and locking a route, first check that none of its conflicting routes are already asserted.* The properties in this class can be described by the following template:

$$G\left(Xr_i\_\mathrm{LR} \Rightarrow \wedge_{j=1}^{n} \neg \mathrm{con}\_r_j\_\mathrm{LR}\right). \qquad (P_1)$$

For example, the control table of Fig. 3 specifies that Route $1A$ has conflicts with Route $C1A$, Route 4, Route $8A$, Route $78A$ (see the first row of the table). Therefore, SafeR, generates the following Linear Temporal Logic (LTL) property using the template shown above:

$$\mathtt{G}\,(\mathtt{X\ \ 1A\_LR} \Rightarrow \neg\mathtt{C1A\_LR} \wedge \neg\mathtt{4\_LR} \wedge \neg\mathtt{8A\_LR} \wedge \neg\mathtt{78A\_LR}).$$

It may be noted that the actual safety requirement is to guarantee that conflicting routes are not allowed to operate at the same time. Stated directly, the property would be:

$$\mathtt{G}\,(\neg(\mathtt{1A\_LR} \wedge (\neg\mathtt{C1A\_LR} \vee \neg\mathtt{4\_LR} \vee \neg\mathtt{8A\_LR} \vee \neg\mathtt{78A\_LR}))).$$

This direct statement will not satisfy the strong safety requirement in a signaling system because the system can be reset to such a state. On the other hand, the property generated by SafeR aims to prove that the system never transits to an unsafe state from a safe state, which is a strong safety requirement that can be proven using a perfect invariant test. In other words, we lift safety properties on states to strong safety requirements on the transition relation.

2) *Prove that all the points within the route, its overlap and isolation are in their specified position and are locked.* From Table I, we observe that a point is moved to normal position by the NCR relay and to reverse position by the RCR relay. Therefore, the following pair of templates are used by SafeR:

$$G\left(Xp_i\_\text{NCR} \Rightarrow \vee_{j=1}^{n}\text{normal}\_r_j\_\text{LR}\right) \qquad (P_{2.a})$$

$$G\left(Xp_i\_\text{RCR} \Rightarrow \vee_{j=1}^{m}\text{reverse}\_r_j\_\text{LR}\right). \qquad (P_{2.b})$$

Here p_i denotes the ith point in the yard layout. The routes setting p_i in normal and reverse are denoted by normal_r and reverse_r respectively. We assume that there are a total of n number of routes that require point p_i to be in normal position and m number of routes require p_i to be in reverse position. From the control table of Fig. 3, we find that routes $1A$, $C1A$, and 4 need point 51 in its normal position and routes 6, 1BD, 1BM, and $C1B$ need 51 to be in its reverse position. For point 51, the properties generated by SafeR are the following:

$$\texttt{G}\,(\texttt{X 51\_NCR} \Rightarrow (\texttt{1A\_LR} \vee \texttt{C1A\_LR} \vee \texttt{4\_LR}))$$
$$\texttt{G}\,(\texttt{X 51\_RCR} \Rightarrow (\texttt{6\_LR} \vee \texttt{1BD\_LR} \vee \texttt{1BM\_LR} \vee \texttt{C1B\_LR})).$$

3) *Prove that the selected route, its overlap and isolation are locked.* ASR relays prove that a route is locked when de-energized. UCR relays prove that all the points in a route, its overlap and isolation are locked. UCR is also known as route checking relay. We use the following template:

$$G(r_i\_\text{UCR} \Rightarrow X\neg r_i\_\text{ASR}). \qquad (P_3)$$

Here r_i denotes ith route of the yard. The Properties generated using this template ensure that a route can only be locked at a particular time instant if route checking is successful at any previous time step. For route $1A$ the property generated is as shown below:

$$\texttt{G}\ (\texttt{1A\_UCR} \Rightarrow \texttt{X}\neg\texttt{1A\_ASR}).$$

4) *Prove that the track circuits within the route are clear up-to the next signal and the track circuits of its overlap are also clear.* We use the following template:

$$G\left(Xs_i\_\text{HR} \Rightarrow \wedge_{j=1}^{n}t_j\_\text{TPR}\right). \qquad (P_4)$$

Here s_i denotes the ith signal of a yard and t_j denote the control tracks of routes originating from s_i. HR relay proves that its corresponding signal is showing a single yellow aspect. The TPR relays prove track occupancy. Properties generated using this template verify that a signal should be cleared (turning YELLOW from RED) only if control-tracks of the route being asserted are clear at the previous time step. Instantiating this template for signal $S6$ generates the following property:

$$\texttt{G}\ (\texttt{X 6HR} \Rightarrow (\texttt{6TPR} \wedge \texttt{4TPR} \wedge \texttt{02TPR})).$$

The route with $S6$ as entry signal is 6. Control-tracks for route 6 are $6T$, $4T$, and $02T$.

5) *Proving that all interlocked level crossing gates are closed and locked against road traffic in the route and overlap, if any.* We use the following template:

$$G\left(Xs_i\_\text{HR} \Rightarrow \wedge_{j=1}^{n}\neg lc_j\_\text{YR}\right). \qquad (P_5)$$

Here lc_j denotes the jth level crossing interlocked with any route which has s_i as its entry signal. The LCYR relays, when energized, indicate that the corresponding level crossing is not closed. So properties generated using this property template assure that a signal can never be cleared when a level crossing is still open. For signal $S1$, the property generated by SaferR is shown below:

$$\texttt{G}\,(\texttt{X 1HR} \Rightarrow \neg\ \texttt{LC1\_YR}).$$

It is clear from the yard layout that to assert any route with its entry signal as $S1$ we need the level crossing LC1 to be closed first. Verifying this property assures the same.

6) *Clear the signal after proving all the conditions mentioned above.* We use the following template:

$$G\left(Xs_i\_\text{HR} \Rightarrow \wedge_{j=1}^{n}\neg p_j\_\text{WLR}\right). \qquad (P_6)$$

As before, s_i and p_j denote the ith signal and the jth point of a route having entry signal as s_i. A WLR relay proves that a point is locked and the locking is detected. This acts as the final check after which the signal can be cleared. Properties generated using this template ensure that a signal is cleared only when point detection is done at the previous time step. For example, the property generated for route $8A$ is shown below:

$$\texttt{G}\,(\texttt{X 8HR} \Rightarrow \neg\texttt{52\_WLR}).$$

Routes with signal 8 as entry-signal are $8A$, 8BD, and 8BM. All these routes are concerned with point 52 only.

It may be noted that all the properties generated by SafeR are properties of the transition relation, namely that they define the admissible transitions of the system. These properties are strong safety requirements, since robust implementations of signaling systems guarantee that transitions from safe to unsafe states are not supported by the transition relation regardless of the state of the system.

As shown in Fig. 1, input to SafeR is the control table of a yard represented in XML format. A java-based XML parser is implemented in SafeR to read through the control table one route at a time and obtain route specific data. The parsed data is then used to instantiate each of the property templates as required. The algorithm implemented in SafeR for automatic formal property generation is shown in Algorithm 1. The main loop runs for all the routes present in the control table. The inner loops run for either the total number of points or for the total number of control tracks or the total number of level crossings for a particular route. Finally, the set of properties $\mathcal{P}$ is returned.

Algorithm 1: *Property generator for route setting / locking*

```
Data: Control table (CT) with routes $\{r_1, r_2, \ldots, r_n\}$
Result: Set of formal properties $\mathcal{P}$
INITIALIZE : $\mathcal{P} = \phi$
// Find the number of points in the yard
$m$ = CountPoints();
for points $p_i : i := 1$ to $m$ do
    // Create two lists for each of the points
    CreateList($p_i Normal$); CreateList($p_i Reverse$);
for routes $r_i := r_1$ to $r_n$ do
    // Find the set of conflicting routes $C$ for $r_i$
    $C$ = ConflictingRoutes($r_i$);
    // Generate properties for template $P_1$
    for all $r_j \in C$ do
        $\mathcal{P} = \mathcal{P} \cup \{$G( X($r_i LR$) $\Rightarrow$ ($\neg r_j LR$))$\}$;
    // Find the overlap for $r_i$
    $r_i OV$ = Overlap($r_i$);
    // Find the set of points for $r_i$ and $r_i OV$
    $P_n$ = PointsNormal($r_i$) $\cup$ PointsNormal($r_i OV$);
    $P_r$ = PointsReverse($r_i$) $\cup$ PointsReverse($r_i OV$);
    // Check for points position for $r_i$ and $r_i OV$
    for all $p_j \in P_n$ do
        Add $r_i LR$ to $p_j Normal$;
    for all $p_j \in P_r$ do
        Add $r_i LR$ to $p_j Reverse$;
    // Generate properties for template $P_3$
    $\mathcal{P} = \mathcal{P} \cup \{$G(($r_i UCR$) $\Rightarrow$ ( X $\neg r_i ASR$))$\}$;
    // Find the set of control-tracks for $r_i$
    $T$ = ControlTracks($r_i$);
    // Find the entry signal for $r_i$
    $S_{entry}$ = EntrySignal($r_i$);
    // Generate properties for template $P_4$
    for all $t_j \in T$ do
        $\mathcal{P} = \mathcal{P} \cup \{$G( X($S_{entry} HR$) $\Rightarrow$ ($t_j\_TPR$))$\}$;
    // Find the set of level-crossings for $r_i$
    $Lc$ = LevelCrossings($r_i$);
    // Generate properties for template $P_5$
    for all $lc_j \in Lc$ do
        $\mathcal{P} = \mathcal{P} \cup \{$G( X($S_{entry} HR$) $\Rightarrow$ ($\neg lc_j YR$))$\}$;
    // Generate properties for template $P_6$
    for all $p_j \in P_r \cup P_n$ do
        $\mathcal{P} = \mathcal{P} \cup \{$G( X($S_{entry} HR$) $\Rightarrow$ ($\neg r_j WLR$))$\}$;
// Generate properties for template $P_{2.a}$
for List $p_i Normal$: $i := 1$ to $m$ do
    for all $r_j LR \in p_i Normal$ do
        consequent = consequent $\vee$ $r_j LR$
    $\mathcal{P} = \mathcal{P} \cup \{$G( X($p_i NCR$) $\Rightarrow$ consequent)$\}$;
// Generate properties for template $P_{2.b}$
for List $p_i Reverse$: $i := 1$ to $m$ do
    for all $r_j LR \in p_i Reverse$ do
        consequent = consequent $\vee$ $r_j LR$
    $\mathcal{P} = \mathcal{P} \cup \{$G( X($p_i RCR$) $\Rightarrow$ consequent)$\}$;
return $\mathcal{P}$;
```

Experimental results obtained from two real case studies are presented in Section VI, where formal properties are generated automatically using the SafeR tool.

IV. The PIPE Prover Engine

The *Perfect Invariant Prover Engine* (PIPE) parses the given application logic of a yard and proves the set of formal properties generated by SafeR for that yard. As is the existing practice, the application logic is given in the form of *ladder logic* over the relays used in the development process.

Each line of code in the application logic represents a single rung of ladder logic. A sample line of code in the application logic is shown below:

$$\text{ASSIGN}$$
$$\sim S1\_\text{UCR} * \sim S1\_\text{HR} *$$
$$(S1\_\text{ASR} + (\sim R1\_\text{LR} * T1\_\text{TPR} * S1\_\text{UYR2} * \sim S1\_\text{TSR}$$
$$* T2\_\text{TPR} * S1\_\text{RECR})) \text{ TO } S1\_\text{ASR}$$

The PIPE tool uses the off-the-shelf model checking tool, NuSMV [18] (at the back-end). The PIPE compiler converts ladder logic into an SMV model which is the input format for NuSMV. The relays used in the application logic are represented by Boolean state variables in the model and each rung of the ladder logic defines the next value assignment for a state variable. For example, for the sample line of code shown above, the SMV model is as follows:

```
MODULE main
VAR
  rS1_UCR: boolean; rS1_HR: boolean;
  rS1_ASR: boolean; rR1_LR: boolean;
  rT1_TPR: boolean; rS1_UYR2: boolean;
  rS1_TSR: boolean; rT2_TPR: boolean;
  rS1_RECR: boolean;
ASSIGN
  next(rS1_ASR):=!rS1_UCR & !rS1_HR &
  (rS1_ASR|(!rR1_LR & rT1_TPR & rS1_UYR2
  & !rS1_TSR & rT2_TPR & rS1_RECR))
```

The proof engine deals with two types of properties. Its primary concerns are with the strong safety properties generated by SafeR. Additionally, if the initial states (or reset states) are given, then the engine must prove the safety of these states. We use bounded model checking (BMC) for both these types, as described in the following example.

Example 4.1: Let us recall the following property described in Section III for proving that the interlocking is free for Route $1A$.

$$\texttt{G}(\texttt{X 1A\_LR} \Rightarrow \neg\texttt{C1A\_LR} \wedge \neg\texttt{4\_LR} \wedge \neg\texttt{8A\_LR} \wedge \neg\texttt{78A\_LR}).$$

Since this is a strong safety requirement, our tool generates the NuSMV model with every state as a potential initial state (that is, the choice of the initial state is non-deterministically chosen from the entire set of states) and runs BMC with bound of 2 to prove the property:

$$(\texttt{X 1A\_LR} \Rightarrow \neg\texttt{C1A\_LR} \wedge \neg\texttt{4\_LR} \wedge \neg\texttt{8A\_LR} \wedge \neg\texttt{78A\_LR}).$$

This proves that the logic does not admit a transition from a safe state to a unsafe state. It does not prove that the initial state is safe, or that the reset states are safe. Now suppose the initial and reset states are given (these are known for any EI system). Then our tool uses the NuSMV model to prove the following property at all initial/reset states:

$$\neg\left(\texttt{1A\_LR} \wedge (\neg\texttt{C1A\_LR} \vee \neg\texttt{4\_LR} \vee \neg\texttt{8A\_LR} \vee \neg\texttt{78A\_LR})\right).$$

This proves that all initial and reset states are safe, and therefore, together with the strong safety property it proves that the system is safe with respect to this signaling principle. □

V. Test Point Prioritization

In current practice, the application logic is tested at two levels, broadly known as *factory acceptance test (FAT)* and *site acceptance test (SAT)*. FAT testing is sometimes done in the premises of the third party developer of the application logic. SAT testing is performed after the application logic has been loaded into the EI equipment and the EI equipment has been interfaced with the relays at the site of its deployment. SAT testing is the last barrier before the equipment is commissioned in the yard.

In this section, we focus on the use of the formal properties generated by SafeR for test prioritization during SAT testing. While the primary goal of FAT testing is to find errors in the application logic, the objective of SAT testing is to verify whether the signaling setup consisting of the actual track side equipment and the EI equipment is working correctly. SAT testing is therefore a time consuming process and may take many hours for small yards and even several days for the larger yards. This is not so much of a problem for a new yard which is not yet operational, but a very significant challenge for a yard which is already in use and is being upgraded.

A yard upgrade may happen in various scales—from adding a few track segments in an existing route to adding several new routes in an existing yard. The yard upgrade is accompanied by regeneration of the application logic and comprehensive SAT testing. However, the main challenge is in the SAT testing, because it may cause severe disruption in railway traffic through that yard.

It is important to realize that the upgrade may not significantly affect the physical assets in most of the existing routes. However some of these routes may be indirectly affected due to their conflicts with the new routes. Our intention is to prioritize the SAT tests in a way that the tests that are directly or indirectly related to the modification have higher priority than those that are not supposed to be affected. Performing these tests early may help in finding the errors due to the upgrade at an early stage of SAT testing.

We achieve test-point prioritization by formalizing the test plan and performing a formal coverage analysis vis-a-vis the properties generated by SafeR. An intuitive definition of test-point coverage is as follows.

Definition 5.1 [Test-Point Coverage of Formal Property]: A test point, T_i, is said to cover a formal property, P, if for all the scenarios in which T_i fails, P fails as well. □

Our *Prioritization Engine for Test-points* (PET) performs the following steps for each test-point, T_i:

- T_i is converted to an LTL formula. Since the set of tests performed are standard, the generation of LTL properties corresponding to test points can be done using templates, in roughly the same way as SafeR generates the formal properties for signaling validation.
- The validity of the formula $(\neg T_i \Rightarrow \neg P)$ is checked using LTL satisfiability methods. If valid, then T_i covers P as given in Definition 5.1.

Following a yard upgrade, we compare the set, S, of formal properties generated by SafeR for the original yard with the set, S', of formal properties generated by SafeR for the modified yard. We define the set, Δ, as the difference between the two sets, that is,

$$\Delta = S' \setminus S.$$

The set Δ consists of those properties that are newly added or are modified due to the changes in the yard. Finally, for each property P_i in Δ we find the test-points that cover P_i and mark them as high-priority. The following example illustrates the idea.

Example 5.1: Consider the following test-point:

$T1$: *A given route is set and locked. It shall remain locked and the signal put back to danger if any of the back lock tracks or overlap tracks fail.*

This test-point can be expressed using the following LTL formula template over the relays:

$$G\left((\vee_i^n \neg t_i\text{\_TPR}) \Rightarrow X \neg s_j\text{\_HR}\right). \qquad (T_1)$$

Here, t_i denotes the ith track of the concerned route originating from signal s_j. If a track-circuit (TPR) is in de-energized state then it indicates that the track is occupied. Therefore this LTL formula specifies that, if any of the track-circuits in the consequent part is in de-energized state then in the next cycle the proceed aspect relay (HR) of the signal should be de-energized. The corresponding formal property for this test is Property P_4 given in Section III.

Now we introduce a minor change in the yard as shown in Fig. 2. We split the track $3T$ into $3AT$ and $3BT$. As a result all such formal properties having `3TPR` as a predicate will get updated. Therefore when we find the difference of sets between the set of properties before and after yard modification, only properties belonging to category P_4 show up. This indicates that all the tests of the above form T_1 must be tagged as high priority. □

The following test-points need to be executed for testing route setting and locking.

1) *Given that a route is set* (ASR *bit cleared and* HR *bit set*), *the route should remain set and locked and signal put back to danger if any of the back lock track or overlap track fail.*

$$G\left((\vee_i^n \; \neg t_i\text{\_TPR}) \Rightarrow X \;\; \neg s_j\text{\_HR}\right) \qquad (T_1)$$

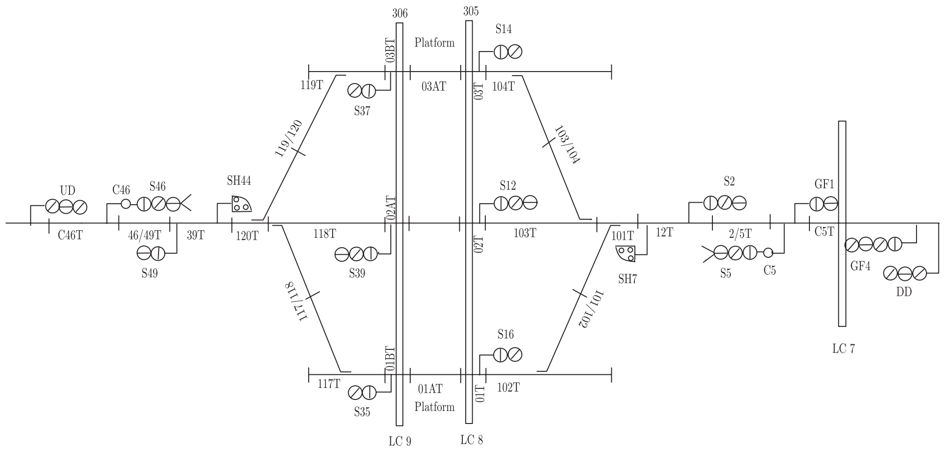


Fig. 4. Dholakpur yard.

where n is the total number of track-circuits combining both the back lock tracks as well as the overlap of the route originating from s_j.

2) *Given that a route is set* (ASR *bit cleared and* HR *bit set*), *the route should remain set and locked and signal put back to danger if point detection is lost.*

$$G\left(\left(\vee_i^n \ \neg p_i\_\text{WLR}\right) \Rightarrow X \ \neg s_j\_\text{HR}\right) \qquad (T_2)$$

where n is the total number of points involved in setting the route s_j.

3) *Given that a route is set* (ASR *bit cleared and* HR *bit set*), *the route should remain set and locked and signal put back to danger if Level crossing lock detection is lost.*

$$G\left(\left(\vee_i^n \ \neg lc_i\_\text{YR}\right) \Rightarrow X \ \neg s_j\_\text{HR}\right) \qquad (T_3)$$

where n is the total number of level crossings intersecting the route s_j.

4) *Under the route locked condition, conflicting routes shall not be available or cannot be set.*

$$G\left(\left(\neg r_i\_\text{ASR} \wedge \neg r_j\_\text{GN}\right) \Rightarrow X \neg r_j\_\text{LR}\right). \qquad (T_4)$$

Here r_i is the route for which the test is generated and r_j is a route conflicting to r_i. The route setting button for r_j is denoted by r_j_GN.

By the proposed coverage analysis we find that test-points T_1, T_2, T_3, and T_4 cover properties P_5, P_6, P_4, and P_1 respectively. Experimental results showing test-points being prioritized based on modifications made to the yard in our case study are presented in Section VI.

VI. Experimental Results

In this section we present our results over two test cases, representing the railway yards of Dholakpur and Gyanpur (actual names obfuscated). The Dholakpur yard is a small yard as shown in Fig. 4. The yard consists of a single bi-directional main line and two bi-directional loop lines. There are 24 routes in this yard. Gyanpur has a much bigger yard having 86 routes (cannot be shown within limited space). Other attributes of these yards are shown in Table II.

TABLE II
Attributes of Dholakpur and Gyanpur Yards

Yard Elements	***Count***	
	Dholakpur	***Gyanpur***
Home Signal	2	4
Starter Signal	6	7
Advanced Starter Signal	2	4
Calling-on Signal	2	4
Shunt Signal	2	7
Route Indicator	4	5
Points	4	14
Track-circuits	22	63
Level-crossings	3	3

TABLE III
Verification Time for Dholakpur and Gyanpur Logic

Template	***Dholakpur***		***Gyanpur***	
	Count	***Time (sec)***	***Count***	***Time (sec)***
P_1	125	33.4	2231	1074.2
P_2	8	4.6	28	10.1
P_3, P_4, P_5, P_6	38	17.3	45	48.6

A. Formal Verification of Application Logic

There are two EI equipments in the Dholakpur yard, whereas the Gyanpur yard has three EI equipments. All the vital and non-vital relays are distributed across these EI equipments. The application logic is developed by third party vendors and made available in textual format in relay ladder logic form for both the yards. We entered the layout of these yards using our layout editor tool and used the built-in compiler to generate the control table. Thereafter our tool, SafeR, was used to generate a total of 171 and 2304 strong safety properties for Dholakpur and Gyanpur respectively.

Our PIPE engine parsed the given application logic for each yard and prepared SMV models. Perfect invariant tests were carried out automatically by PIPE on these SMV models for

TABLE IV
LIST OF PROPERTIES DETECTING THE INJECTED BUGS

Bug Scenario	***Property Detecting the Bug***
F_1	`LTLSPEC G ((!rC46_03-INT_LR-C2 | !r5_03LR-C2C1-C2) → (X rC46_03-INT_LR-C2 → !r5_03LRC2C1-C2))`
F_2	`LTLSPEC G ( X r101_102RCR-C2 → (r5_01LR-C2C1-C2 | r16_12LR-C2C1-C2 | r16_OV1LR-C2 | r16_OV2LR-C2 | r12_OVLR-C2 | r14_OV1LR-C2 | r14_OV2LR-C2 | rC5_01LR-C2 | rSH7_01LR-C2C1-C2))`
F_3	`LTLSPEC G (( r12UCR-C2 ) → X (!r12ASR-C2C1-C2))`
F_4	`LTLSPEC G (X X r5HR2-C2C1-C2 → (r2_5TPR-C2C1-C2 & r12TPR-C2C1-C2 & r101TPR-C2C1-C2 & r102TPR-C2C1-C2 & r01_01A_01BTPR-C2C1-C2 & r103TPR-C2C1-C2) | (r2_5TPR-C2C1-C2 & r12TPR-C2C1-C2 & r101TPR-C2C1-C2 & r103TPR-C2C1-C2 & r02_02ATPR-C2C1-C2) | (r2_5TPR-C2C1-C2 & r12TPR-C2C1-C2 & r101TPR-C2C1-C2 & r103TPR-C2C1-C2 & r104TPR-C2C1-C2))`
F_5	`LTLSPEC G (X rC46HR-C2C1-C2 → !rLC8_YR-C2C1-C2 & !rLC9_YR-C2C1-C2)`
F_6	`LTLSPEC G ( X X r16HR2-C2C1-C2 -> !r101_102WLR-C2C1-C2)`

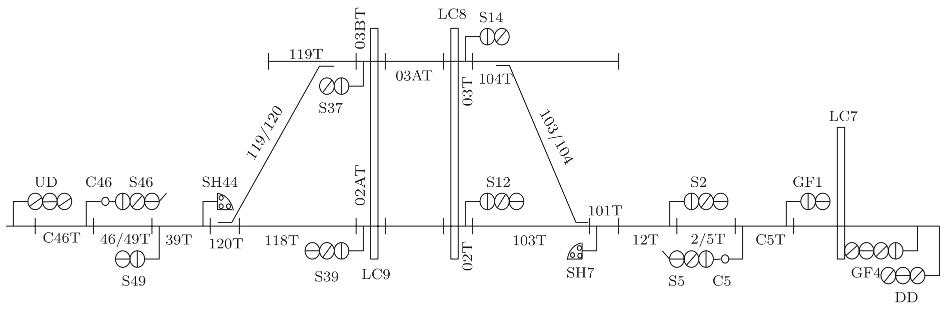


Fig. 5. Yard with a single-loop line.

the entire set of properties generated by SafeR (using the NuSMV tool at the back end) for each yard. The execution times for verifying the properties are given in Table III. All execution times are reported in seconds on a 3 GHz INTEL(R) CORE(TM) i5 processor with 4 GB RAM.

It can be seen from Table III that the total time to verify all 171 properties for Dholakpur yard is less than a minute. This is significantly faster than the time required for FAT testing using present practices. For Gyanpur the total verification time is around 20 minutes for 2231 properties. This is notably much more than that of Dholakpur, but mainly caused by the larger number of properties that had to be verified. The tool did not run into any scalability issue.

The bug reports obtained by our tool on the real yards cannot be shared in this paper due to the sensitive nature of the information. In order to demonstrate that the tool actually finds errors in the application logic, we present a set of representative bug scenarios which have been purposefully injected into the Dholakpur yard logic and then the PIPE tool has been used to find the bugs. The properties that detect these bugs are listed in Table IV. These properties are written as per the input format of NuSMV.

1) ***Bug Scenario*** F_1: *The relay* 5_03LR *is removed from the implementation logic of* 46_03LR. Since Route 5_03 is a conflicting route of 46_03, PIPE detects a violation of property.
2) *Bug Scenario* F_2: *The relay* 46_02LR *is mistakenly inserted in the implementation logic for* 101_102RCR. Since route 46_02 does not require point 101_102 in reverse position, as per the signaling plan, PIPE detects a bug.
3) Bug Scenario F_3: *From the implementation logic of* 12ASR, 12UCR *is removed*. PIPE detects this bug.
4) Bug Scenario F_4: *The track relay* 02TPR *is removed from the implementation logic of* 5HR. This scenario is similar to the Milton Keynes incident [2]. PIPE detects this bug.
5) Bug Scenario F_5: *The status of the level crossing gate* LC8 *is removed from the implementation logic of* C46HR. PIPE detects this bug.
6) Bug Scenario F_6: The HR circuit checks if all the concerned points are set and locked before changing the aspect of a signal. *The implementation* logic of 16HR has been modified such that it gets energized when the point 101_102 point is not locked, by changing the logic state of 101_102WLR in the implementation logic. PIPE detects this bug.

B. Test-Point Prioritization

As discussed in Section V, we prioritize the test points for SAT testing following the upgrade of a yard to minimize the critical downtime for the yard. In order to demonstrate the proposed PET tool flow, we use the simple yard shown in Fig. 5. The yard has 16 routes. For this yard, SafeR generated 110 strong safety properties, of which 74 are route specific, 4 are point specific, and 32 are signal specific properties. We refer to this set of 110 properties as S. The application logic for this yard was formally verified using

TABLE V
AFFECTED PROPERTIES FOR ROUTE ADDITION

Templates		P_1	P_2	P_3	P_4	P_5	P_6
U_1	***Unaffected***	74	0	8	10	4	10
	Partially Updated	0	0	0	0	0	0
	New	51	4	2	2	2	2
U_2	***Unaffected***	125	4	10	5	6	1
	Partially Updated	0	4	0	5	0	11
	New	0	0	0	0	0	0

PIPE. Thereafter we considered the following modifications to this yard.

1) $U1$: *Addition of a new route*. We added another loop line in the yard of Fig. 5 to obtain the yard of Fig. 4. It may be recalled that the yard of Fig. 4 has 24 routes for which SafeR generates 171 properties. Let us call this set of 171 properties as S'. Our PET tool flow starts by computing the difference $S' \setminus S$. In this case, we find that $S' \setminus S$ consists of 67 properties, of which 63 are new and the remaining 4 are existing properties that have been modified due to upgrade. The distribution of these properties is shown in Table V. PET then interprets the test templates to generate the formal test points, and performs the coverage analysis by the test points of the properties in $S' \setminus S$. In this case, PET generated 120 test points. Coverage analysis showed that 67 test points are covered. These test points are marked as high priority.
2) $U2$: *Splitting of track segments*: In this scenario we consider a minor modification made to the yard where the track-circuit 101T is split into two track-circuits, named 101AT and 101BT. Following a similar approach, we again find the new set of properties ($\mathcal{S}'$). The total number of properties generated in this case remains the same but some properties get updated due to the modifications made to the yard. The count of the affected properties is shown in Table V.

The runtimes for the test prioritization steps were found to be quite modest. For U_1, our tool returned in 3.6 sec, and for U_2 it returned in 2.3 sec.

VII. CONCLUSION

Formal verification of the application logic in railway signaling is a highly significant step, given that errors in the application logic are known to have escaped SAT and FAT testing. The focus of our work is to propose a tool flow that supplements the existing practice of using third party support in creating the application logic. Since our tool flow deals with equipment specific application logic, we deal with specific combinations of relays whose semantics is built into the tool. We also use the notion of strong safety properties and the theory of perfect invariants to perform the validation using bounded model checking with known sequential depth. This allows our tool to scale to large yards, though we are not able to share results on such yards.

We believe that using automatic monitors for the properties generated by SafeR can significantly aid the FAT testing procedure. To the best of our knowledge, this is not used by any railway company in their current practices.

Use of formal methods for test-point prioritization is proposed here for the first time. It is difficult to quantify the benefits of test-point prioritization in absolute terms. This is because SAT testing practices are widely different, ranging from shutting down a yard completely for a period of time to manually operating the passage of trains through some routes while the other routes are being tested. However it is generally accepted that test-point prioritization helps in scheduling the SAT testing in a better way and also in putting the best resources for the high priority test points.

APPENDIX

Different signaling terms used in Fig. 3 and Table II are defined here.

1) *Entry Signal*: The first signal in the route.
2) *Exit Signal*: The last signal in the route.
3) *Point Normal/Reverse*: The orientation of points required for setting a particular route.
4) *Overlap*: The set of track-segments starting from an exit signal of a route up to the next signal.
5) *Overlap Normal/Reverse*: The orientation of points required for setting the overlap of a particular route.
6) *Signal Replaced by Track*: The track(s) whore occupation should turn the entry signal of a route from CLEAR to DANGER (RED aspect).
7) *Back-locked Track*: The continuous set of tracks from the start of the route up to the last track connected to a point.
8) *Home Signal*: This is the first stop signal on approach to a yard. This signal guards the entry to a yard and is placed before all the branches (loop lines) connected to the main line.
9) *Starter Signal*: This governs exit from a yard. It is placed at the most advanced section of a yard up to which a train is allowed to move.
10) *Advance Starter Signal*: It is a stop signal provided ahead of the starter signal. If present it is the last stop signal on departing yard limits.

REFERENCES

[1] A. Fantechi, W. Fokkink, and A. Morzenti, *Some Trends in Formal Methods Applications to Railway Signaling*. Hoboken, NJ, USA: Wiley, 2012, pp. 61–84.
[2] *Raib Review of the Railway Industry's Investigation of an Irregular Signal Sequence at Milton Keynes*, Dec. 29, 2008. [Online]. Available: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/411158/101223-SI2010-MiltonKeynes.pdf
[3] *Reported Signal Irregularity at Cootamundra New South Wales Involving Trains st22 and 4mb7*. [Online]. Available: https://goo.gl/DLxhgg
[4] J. Boulanger, *CENELEC 50128 and IEC 62279 Standards*. Hoboken, NJ, USA: Wiley, 2015.
[5] B. Fringuelli, E. Lamma, P. Mello, and G. Santocchia, "Knowledge-based technology for controlling railway stations," *IEEE Expert*, vol. 7, no. 6, pp. 45–52, Dec. 1992.
[6] G. LeGof, "Using synchronous languages for interlocking," in *Proc. 1st Int. Conf. Comput. Appl. Transp. Syst.*, 1996.
[7] C. Bernardeschi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi, and D. Romano, "A formal verification environment for railway signaling system design," *Formal Methods Syst. Des.*, vol. 12, no. 2, pp. 139–161, 1998.
[8] A. Cimatti, F. Giunchiglia, G. Mongardi, D. Romano, F. Torielli, and P. Traverso, "Model checking safety critical software with spin: An application to a railway interlocking system," *Computer Safety, Reliability and Security*. Berlin, Germany: Springer-Verlag, 1997.

[9] C. Eisner, “Using symbolic CTL model checking to verify the railway stations of Hoorn-Kersenboogerd and Heerhugowaard,” *Int. J. Softw. Tools Technol. Transfer*, vol. 4, no. 1, pp. 107–124, 2002.
[10] M. J. Morley, “Safety in railway signalling data: A behavioural analysis,” in *Higher Order Logic Theorem Proving and its Applications*. New York, NY, USA: Springer-Verlag, 1994, pp. 464–474.
[11] A. Haxthausen, “Automated generation of formal safety conditions from railway interlocking tables,” *Int. J. Softw. Tools Technol. Transfer*, vol. 16, no. 6, pp. 713–726, 2014.
[12] “Applied bounded model checking for interlocking system designs,” in *Software Engineering and Formal Methods*, vol. 8368, ser. Lecture Notes in Computer Science, S. Counsell and M. Nez, Eds. New York, NY, USA: Springer, 2014.
[13] P. Dasgupta, M. K. Srivas, and R. Mukherjee, “Formal hardware/software co-verification of embedded power controllers,” *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 33, no. 12, pp. 2025–2029, Dec. 2014.
[14] A. Pnueli, “The temporal logic of programs,” in *Proc. 18th Annu. SFCS*, Washington, DC, USA, 1977, pp. 46–57.
[15] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, *Symbolic Model Checking Without BDDs*. New York, NY, USA: Springer-Verlag, 1999.
[16] M. Sheeran, S. Singh, and G. Stålmarck, “Checking safety properties using induction and a sat-solver,” in *Proc. 3rd Int. Conf. FMCAD*, 2000, pp. 108–125.
[17] *Indian Railway Standard Specification for Relay Interlocking Systems*, RDSO, Indian Railways, S36-87-ris.
[18] A. Cimatti *et al.*, “NuSMV 2: An opensource tool for symbolic model checking,” in *Computer Aided Verification*. Berlin, Germany: Springer-Verlag, 2002, pp. 359–364.

Shiladitya Ghosh received the B.Tech. degree in computer science and engineering from St. Thomas' College of Engineering and Technology, Kolkata, India. He is currently working toward the M.S. degree (by research) in computer science and engineering from Indian Institute of Technology Kharagpur, Kharagpur, India.

Arindam Das received the B.Tech. degree in instrumentation engineering from Haldia Institute of Technology, Haldia, India.

He is a Senior Research Fellow with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur, India.

Nirvik Basak received the B.Tech. degree in computer science and engineering from B.P. Poddar Institute of Management and Technology, Kolkata, India, and the M.Tech. degree in computer science and engineering from Indian Institute of Technology Kharagpur, Kharagpur, India.

Pallab Dasgupta received the B.Tech, M.Tech, and Ph.D. degrees in Computer Science and Engineering from Indian Institute of Technology Kharapur in 1990, 1992, and 1995, respectively.

He is a Professor with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur, India, where he is an Associate Dean with the Centre for Sponsored Research & Industrial Consultancy. His research interests include formal verification and computer-aided drafting.

Prof. Dasgupta is a Fellow with the Indian National Academy of Engineering and the Indian Academy of Science.

Alok Katiyar received the B.Tech degree in Electronics from Institute of Engineering and Technology Lucknow in 1990.

He is a Director/Signal with the Research Design and Standards Organization, Indian Railways, Lucknow, India. His expertise is in metro signaling systems, high-speed trains, electronic interlocking, data acquisition systems, electromagnetic interference/electromagnetic compatibility, software-embedded critical safety systems, product development for high-tech signaling products, and systems for Indian railways. He has more than 21 years of experience in design, installation, commissioning of high-tech signaling systems, system planning, and time-bound implementation of critical metro and main line projects.