Deductive Verification of Railway Operations

Eduard Kamburjan and Reiner Hähnle

Department of Computer Science, Technische Universität Darmstadt, Germany
{kamburjan,haehnle}@cs.tu-darmstadt.de

Abstract. We use deductive verification to show safety properties for the railway operations of Deutsche Bahn. We formalize and verify safety properties for a precise, comprehensive model of operational procedures as specified in the rule books, *independently* of the shape and size of the actual network layout and the number or schedule of trains. We decompose a global safety property into local properties as well as compositionality and well-formedness assumptions. Then we map local state-based safety properties into history-based properties that can be proven with a high degree of automation using deductive verification. We illustrate our methodology with the proof that for any well-formed infrastructure operating according to the regulations of Deutsche Bahn the following safety property holds: whenever a train leaves a station, the next section is free and no other train on the same line runs in the opposite direction.

1 Introduction

In the paper [14] we reported on our ongoing effort to create a formal and highly comprehensive model of the regulations described in the rulebooks [4,5] that govern railway operations of Deutsche Bahn. This executable model is expressed in terms of the Abstract Behavioral Specification (ABS) language [12], a formal, concurrent modeling language that follows the active objects paradigm. ABS is equipped with a program logic that supports specification and verification of properties expressed over first-order event histories. The program verification system KeY-ABS [6] allows users to perform mechanical proofs of safety properties for ABS models by means of deductive verification [1]. In [14] we gave a proof-of-concept that deductive verification of safety properties for our ABS railway model is possible. The main contribution of the present paper is to extend that approach into a full-fledged *verification methodology* for railway operations.

Rulebooks are long and complex documents that—at their core—describe those communication protocols between train drivers, controllers, track elements, etc., that are supposed to guarantee safe operation. Their complexity stems mainly from the requirement to ensure continuing and safe train operation even in the case of failure of individual components. Moreover, at any time the system must guarantee that any safety-critical action cannot be inadvertently revoked or compromised. Changing a rulebook and having it re-certified is a complex, time-consuming, and expensive procedure, for which at the moment only minimal tool support and no formal analysis is available. For this reason, a methodology for

the formalization and tool-supported verification of safety properties pertaining to rulebooks is highly desirable.

Rulebooks of railway operators state operational rules that are valid for *any* track layout that satisfies certain regulations [5], *well-formed infrastructures*, as we call them. The rules are also valid independent of the number of trains or schedules, as long as these satisfy a valid initial state (for example, not more than one train is placed in each segment). This means that model checking is ruled out as a technique for verification of *global*, system-wide properties. Instead, we use deductive verification in the program logic of ABS, as outlined in Fig. 1.

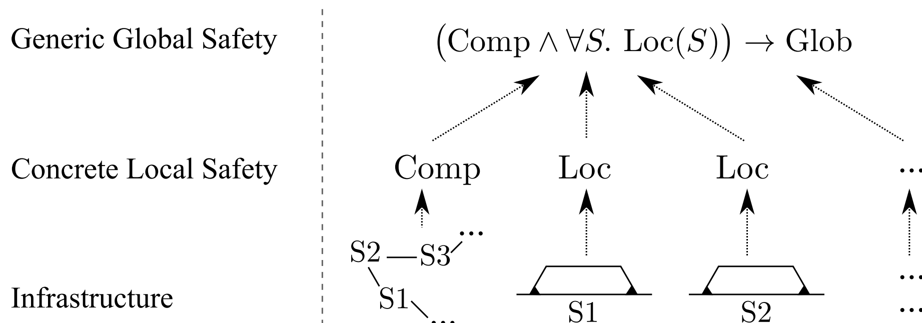

Fig. 1: Decomposition of global safety properties. Loc are local guarantees, Comp composition guarantees, Glob a global safety property. Si are stations.

Assume we want to prove that a global safety property Glob holds in a given ABS model for any well-formed infrastructure, any number of trains and any number of stations S. First, we decompose the proof of this property into proofs for a local guarantee Loc at each station and composition guarantees Comp. For local composition guarantees (for example, aspects of the interlocking system), established model checking techniques may be used. Our approach is not intended to replace established and well-working verification technology, but to *extend* it so as to be able to prove *global* properties of a highly *precise* model. Local guarantees Loc only hold under assumption of a well-formed infrastructure, expressed in Comp.

In this paper we make two contributions: first, a systematic methodological approach to decompose global system properties for any well-formed infrastructure into local guarantees that are then proven by a combination of deductive verification and model checking. As detailed in Sect. 5.1, we prove state-based global properties expressed over actions by transforming them into history-based properties of processes. The latter can then be expressed and proved as local method invariants in KeY-ABS. Second, we demonstrate the viability of our approach with an ABS model of a part of the actual Deutsche Bahn rulebooks and a typical safety property. Contrary to prior work [14] we prove (1) the procedures

used in current operations and (2) show a global safety property, rather than only one of its fractions.

The paper is organized as follows: In Sect. 2 we explain very briefly those elements of the ABS language needed to understand the paper. In Sect. 3 we give a short account of the program logic of ABS. Sect. 4 explains how we modeled railway operations and rules in ABS. Sect. 5 is the core of the paper where we explain our methodology in detail and sketch the proof of one of the safety properties. We conclude, and give related as well as future work, in Sect. 6.

2 The Abstract Behavioral Specification Language (ABS)

We give a very brief introduction to the Abstract Behavioral Specification language (ABS), for a full account and the formal semantics we refer to [10,12].

ABS is a modeling language for distributed systems, which has been designed with a focus on analyzability. Its syntax and semantics are similar to Java to maximize usability. We list the main language features (slightly simplified) and the statements associated with them.

Strictly encapsulated objects. Communication between different objects is only possible via method calls. All fields of an object are private and inaccessible even to other instances of the same class and there are no static fields. This ensures that the heap of an object is only accessed by its own processes.

Asynchronous communication with futures. Asynchronous calls are dispatched with the statement `Fut<T> f = o!m(e)`, where method `m` is called on the object stored in `o` with parameters `e`. Upon making this call, the caller obtains the *future* `f` and continues execution without interrupt . A future is a handle to the called process and may be passed around. Once the called process terminates, its return value may be accessed via the associated future. To read a value from a future, the statement `T i = f.get;` is used.

Cooperative scheduling. In ABS at most one process is active per object. Running processes cannot be preempted, but give up control only when they suspend or terminate. Hence the ABS modeler has explicit control over interleaving. The active process suspends itself by waiting for a guard. A guard can be a future—then the suspension statement has the form `await f?;` and the process may become active again once `f` was resolved (i.e., its process terminated). Otherwise, a guard can be a side-effect-free Boolean expression—then the suspension statement has the form `await e;` and the process may become active again if `e` evaluates to true. If a future is accessed with `f.get` before it was resolved, then the whole object blocks until `f` is resolved. When blocked, an object may still receive method calls, but it will not execute them.

Cooperative scheduling enables one to reason about code between the start and end of a method, as well as suspension statements, as if it were executed sequentially, because the process is guaranteed to have exclusive access to the memory of its object. ABS is not completely object-oriented, as the enforced

asynchronous communication leads to overhead for simple look-up operations. To avoid the overhead, ABS uses Algebraic Data Types (ADT) to abstract from data values which have no internal state. Figure 4 shows an ABS class using the ADT `SignalState`.

3 The ABS Program Logic

The calculus used for reasoning about concurrency in ABS uses a *history* of *communication events* [6,7], modeled as finite first-order sequences. A communication event is an action on a future: either an *invocation* event modeling an asynchronous method call, an *invocation reaction* event, modeling the start of the corresponding process, a *completion* event modeling the termination of a process, and a *completion reaction* event modeling the read access to a future.

Definition 1 (Events). *Let o, o' range over object IDs, f over futures, e over values and m over method names. The symbol e^* denotes a possibly empty sequence of values and represents the parameters of a method call. Events* Ev *are defined by the following grammar:*

$$\begin{aligned}
\mathsf{Ev} ::= \;& \mathsf{invEv}(o, o', m, f, e^*) && \textit{(Invocation Event)}\\
\mid\;& \mathsf{invREv}(o, o', m, f, e^*) && \textit{(Invocation Reaction Event)}\\
\mid\;& \mathsf{futEv}(o', m, f, e) && \textit{(Completion Event)}\\
\mid\;& \mathsf{futREv}(o, f, e) && \textit{(Completion Reaction Event)}
\end{aligned}$$

Histories are used for a compact representation and specification of communication behavior. They abstract away from computations and allow to reason directly about communication on futures.

Figure 2 illustrates the connection of events to processes and futures. Every history h, which an ABS system produces is *well-formed*, satisfying certain conditions on the ordering of events. For example, if there is an $i \in \mathbb{N}$ with $h[i] = \mathsf{invREv}(o, o', f, m, e^*)$, then there must be a $j < i$ with $h[j] = \mathsf{invEv}(o, o', f, m, e^*)$. This condition expresses that every process starts its execution only after it was called. The well-formedness conditions for all event types are listed in [7].

ABS uses invariant reasoning: Safety and consistency properties are formulated as first-order formulas and are shown to hold at the beginning of each method execution and at every suspension point. First-order properties are expressed in the ABS Dynamic Logic (ABSDL) [6], a program logic over statements from the ABS language. Matching the ABS concurrency model, formulas can only access the fields of a single class, hence only reason about a single object. Heap memory is modeled with a dedicated program variable *heap*, which can be accessed and changed with `select` and `store` functions, respectively. While every object has its own heap, multiple heap may be used for technical reasons, e.g., to refer to the state before the method starts.

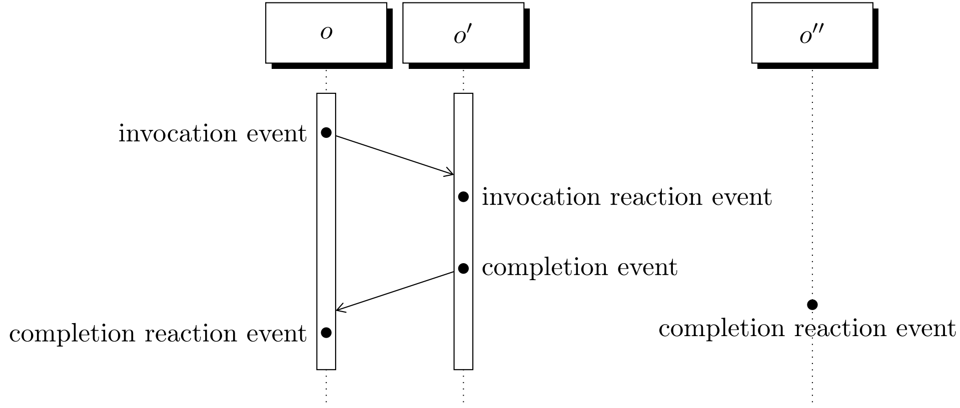

Fig. 2: Events on futures, diagram taken from [6].

Example 1. The following formula ϕ expresses that the field `l` on an object **self** is a list containing only positive values:

$$\phi \equiv \forall\ \mathsf{Int}\ k.\ \big(0 \leq k < \mathit{length}(\mathit{select}(\mathit{heap}, \mathbf{self}, \mathtt{l})) \rightarrow \mathit{select}(\mathit{heap}, \mathbf{self}, \mathtt{l})[k] > 0\big)$$

ABSDL formulas may reference the history visible to the object in question. Whether an ABSDL formula is an invariant for the methods declared in a class can be mechanically checked with the KeY-ABS theorem prover [6].

Example 2. The following ABSDL formula expresses that if the last element of the history h is a completion event on `m`, then ϕ holds

$$\forall\, \mathbf{Fut}\ f.\ \forall\, \mathtt{Object}\ e.\ \mathit{last}(h) \doteq \mathsf{futEv}(\mathbf{self}, \mathtt{m}, f, e) \rightarrow \phi$$

ABSDL formulas are inherently local. To specify global properties we use an extension of ABSDL, called ABSDL*, that lifts the restriction to reason about only one object. Therefore, KeY-ABS cannot be used to reason about ABSDL* formulas. To express that the state σ of an object o satisfies the formula ψ at the moment when the i-th event was added to the history we use the notation $\sigma[i](o) \models \psi$.

Example 3. The following ABSDL* formula expresses that whenever object o reads from a future f, then its field k is positive:

$$\forall\, \mathtt{Object}\, o. \forall m, f, e, i. \Big(h[i] = \mathsf{futREv}(o, f, e) \rightarrow \big(\sigma[i](o) \models \mathit{select}(\mathit{heap}, o, k) > 0\big)\Big)$$

4 Modeling Railway Operations

We give a brief summary of our ABS model for railway operations. Here we are concerned with communication between stations, so we introduce only the most important concepts needed for our safety analysis. In particular, we refrain from describing our train model. The description of the full ABS model is in [14].

4.1 Infrastructure

The concurrency model of ABS is a good match for railway operations: All elements are encapsulated and have no shared memory. Thus all communication can be reduced to message passing, which in turn can be mapped to ABS asynchronous method calls. This unifies the treatment of communication, as we abstract away from the *means* of communication and only consider the communicated *information*.

The railway infrastructure is modeled as a graph, centered around the concept of *point of information flow (PIF)*.

Definition 2 (Point of Information Flow). *A* point of information flow *(PIF) is an object at a fixed position on a track that participates in information flow, if one of the following criteria applies:*

- *It is a structural element allowing a train to receive information, for example, a signal or a data transmission point of a train protection system.*
- *It has a critical distance before another PIF, where its information is transmitted at the latest. E.g., at the point where a signal is seen at the latest.*
- *It is a structural element allowing a train to send information, for example, a track clearance detection device (axle counter), or the endpoints of switches that may transfer information when passed over.*

A PIF is a position at a track and an object that describes the information to be transmitted or relayed. Instead of modeling all features of a PIF in one object, we use a model of four layers to organize and separate its structure:

1. **Graph Layer.** The lowest layer is an undirected graph, where edges correspond to tracks and nodes correspond to the *position* on a track of a PIF. We refer to the set of tracks between two signals as a *section* and to the set of sections between the exit signal of one station and the entry signal of another as a *line*. There may be multiple lines between two stations.
2. **Physical Element Layer.** The second layer corresponds to track elements. Each track element is either a physical device that allows information flow either from or to a *virtual element* that is responsible to model information flow at a specific distance from a physical element. Each element of this layer is assigned to one node of the graph layer. In case several devices are at the same position, a node at the graph layer has multiple track elements assigned.
3. **Logical Element Layer.** The third layer groups physical elements from the second layer, e.g., a pre-signal and a main signal (an exhaustive list of physical elements belonging to a logical object is in [14]). Each physical element can be assigned to multiple logical elements, e.g., a pre-signal can be assigned to two logical signals with two different main signals, or to no logical element, if the physical element never changes its state.
4. **Interlocking Layer.** This layer models the interlocking logic and the communication between stations. Each logical element is assigned to one station.

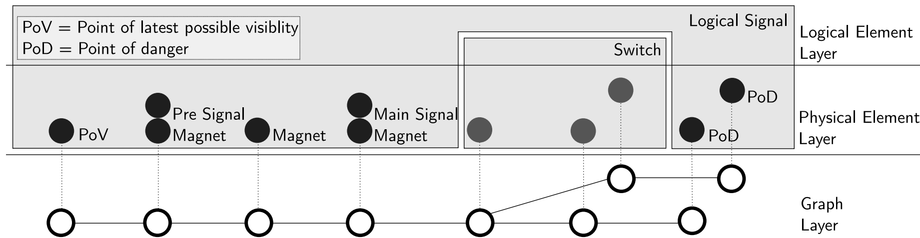

Fig. 3: Illustration of the lower three layers of a station entry in the layer model.

```
1 class SignalImpl(...) implements Signal {
2   SignalState state = STOP;
3   ...
4   Train observedBy = null;
5   Unit triggered() { if (resp != null) resp!triggered(this); }
6   Unit setObserver(Train obs) { observedBy = obs; }
7   Unit setGo() {
8     // ... notify physical elements
9     state = FAHRT;
10    if (observedBy != null) { observedBy!notify(Info(FAHRT),now()); }
11  }
12 }
```

Fig. 4: Simplified implementation of a logical signal.

The lower three layers of a station entry are illustrated in Figure 3. The (simplified) implementation of a logical signal is shown in Figure 4.

The lower three layers only communicate up or down. This means that logical objects only communicate to assigned physical objects and to the assigned station at the interlocking layer. Every global property is established by communication on the interlocking layer.

4.2 Communication

The German railway system has different modes of operation for driving trains outside and inside of stations. Here, we focus on operation outside of stations and do not model, e.g., intermediate signals inside of a station. The rulebooks differentiate between two kinds of stations: *Blockstellen* which operate block signals and only divide a track line into two parts to increase the possible number of trains on the line and *Zugmeldestellen* (Zmst) which are able to "store" trains and rearrange their sequence. The generalization of both is *Zugfolgestelle* (Zfst). In the following, we use the term "station" for Zmst.

A Zfst is responsible for safety on the next section, a Zmst is additionally responsible for establishing safety on a whole line. To let a train drive from Zmst A to Zmst B on a line L, the following conditions must be fulfilled:

- It is possible to set the signal at A covering the first section S of L to "Go", i.e., S is not locked by A and A has the permit token for S.
- B accepted the train and is thus notified about its departure.

There are three communication protocols that ensure safety:

Locking sections. Each Zfst is responsible for several logical elements such as switches and signals. In addition to the internal state of the signals, the interlocking system itself has a state that depends on the neighboring Zfst. Each section has an additional Boolean state *locked*. Consider a signal covering a section leading out of the Zfst. After a signal is set to "Go" and a train passes it, the section it covers is automatically *locked* and the electronic message "*preblock*" is sent to the subsequent signal. A signal cannot be set to "Go" again, as long as the section it covers is locked. It must be unlocked by receiving the "*backlock*" message from the subsequent signal. That signal in turn can only send "*backlock*" after the train passed.

Permit token. For each line there is exactly one token that allows a station to admit trains on this line. Without the token the signal that covers the track cannot be set to "Go".

Accepting and reporting back trains. Before a train leaves a station A with destination B, A *offers* the train and waits for B to accept. This ensures that B has (or will have) a track to park the train. Before the train departs, the departure is *announced* to B. The offering, announcement and acceptance of trains are modeled as methods—the current state of a Zmst is not encoded only in its fields, but also in the currently active (but possible suspended) processes.

4.3 Well-Formedness

The interlocking layer in Section 4.1 only communicates to logical objects, it has neither direct control nor knowledge about the layers below it. Every Zmst is assuming that its knowledge about the train network is correct and that its fields reflect its state correctly. Consider, e.g., an entrance signal: A station is only notified that the train detection device covering the danger point of this signal was triggered. It relies on the guarantees that (1) the train detection device is set up at the correct position and in the correct direction, (2) the train detection device is assigned to the correct signal and (3) the line covered by the signal is indeed the one which is listed as covered inside the station. The most critical point for this to work is the correct encoding of tracks: The `other` field must realize the mapping between the endparts of a line correctly and the line must correspond to a path in the graph layer.

Definition 3 (Well-Formed Infrastructure). *We say that an ABS railway model is* well-formed*, when its initialization block fulfills the following conditions:*

Correct Encoding of Lines. *Every line corresponds to a path through the graph of the lowest layer and is partitioned correctly into sections according to the intermediate signals. The* `other` *field of the Zmst implementation realizes lines correctly: If a line L has starting sections S, S' then for the neighbouring stations map the section on each other: $S = \texttt{other}(S')$ and $S' = \texttt{other}(S)$. Formally, if L is a line between two stations A and B with starting sections S, S' then the following holds:*

$$S = A.\texttt{other}(S') = B.\texttt{other}(S') \qquad S' = A.\texttt{other}(S) = B.\texttt{other}(S)$$
$$A.\texttt{other}(S'') = S \rightarrow S'' = S' \qquad B.\texttt{other}(S'') = S \rightarrow S'' = S'$$
$$A.\texttt{other}(S'') = S' \rightarrow S'' = S \qquad B.\texttt{other}(S'') = S' \rightarrow S'' = S$$

Additionally, for each line L, the method `forcePermit`*, which initializes the permit token, is called exactly once.*

Correct Encoding of Zfst. *For each section S bordering a Zfst, the field* `next` *encodes sections correctly. I.e.,* `next`*(S) is the signal at the end of S. For each signal S', we denote its covered section with S'*`.covers`.

This definition of well-formedness is suitable for our verification methodology and can be extended. For example, we do not reason about safety inside the stations here, but a fitting well-formedness condition would be the classical notion of safety for interlocking systems. Well-formedness is decidable, but here we are not concerned with checking an initialization block for well-formedness.

5 Deductive Verification

5.1 Methodology

Safety properties in technical documents, e.g. [18] are given as informal descriptions. A system state is considered safe if it fulfills a property. ABS is verified with invariants, which state that the history, i.e., the *past* states have a certain property. To express safety properties of railways we connect *state* invariants with *history* invariants. As described in Section 4, we map railway concepts partly to methods instead of fields. E.g., the dispatching of a train is modeled by the method `process`. In a well-formed infrastructure, we can connect events of methods with the state of the whole system. To model informally stated safety notions in ABSDL we use the following schema:

1. We formulate the safety notion *informally* as a property of the global state.
2. We reformulate the safety notion *informally* as a property of *past actions*.
3. Using the model in Section 4, we map actions to methods and states to fields, thus deriving a formal, global invariant of histories in ABSDL*.
4. Finally, we prove the global invariant by splitting it into local invariants by using well-formedness of histories and infrastructure. To connect histories and state, we formulate and prove local invariants with KeY-ABS.

Well-formedness of the infrastructure is needed at two points: In step 3 it is used to connect model and reality: E.g., only in a well-formed infrastructure we can assume that the termination of `process(t)` models a dispatch of train `t` and does not set the route and signal for some other train: we need well-formedness to translate the informal property "*A train* `t` *was dispatched.*" to "*Method* `process (t)` *terminated.*". In step 4 well-formedness is used to reason about consistency of the model: E.g., only in a well-formed infrastructure we can assume that a signal `S` covering line `L` is indeed unlocked by the signal at the end of `L`.

We illustrate deductive reasoning with two safety properties. Each of the property establishes (partial) safety on one of the described layers for train departure: The first property establishes that the permit token is exchanged correctly and the second property establishes that a signal is set to "Go" only when the covered section is free. Together with the obvious property that for every line at any given point in time only one station has the token [1], we regard these properties as the safety notion for departure of trains from A to B: the next section is free and the whole line is free of trains going from B to A. For presentation's sake, we only present the proof of the first property in detail.

5.2 Permission

Recall the description of the permit token from Section 4: Each line L has an associated token. This token models the permission to dispatch trains on this line. The token is implemented as a field `permit` in the Zmst class that maps the first section of the line to a Boolean value modeling the token. A Zmst has the token for a section st if `permit`$[st]$ is set to `True`. When a station plans to dispatch a train, it must first acquire the token for line L. The exchange is not only secured by the station having the token, but also by the station requesting it: The requesting station knows which trains are on the line in its direction, as all the trains are announced and saved as expected. It only requests the token if it is known that no trains are on the line in its direction. The station having the token only checks that the token is not locked, i.e., it is not in the process of dispatching a train using this token.

We examined the case where only the station having the token secures it in [14], however, that protocol is not in use in modern railway operations of Deutsche Bahn. Here we show the following, more complex, property. This property corresponds to Step 1 in the verification scheme.

> *"If station A acquires the permit token for line L from station B, then there is no train on L with arrival station A."*

Station A acquires the permit token when the call on B.`rqPerm` from method `setPreconditions` terminates. If we assume that all stations are connected correctly, the condition that there are no trains on L with arrival station A can

[1] We do not give a proof for this, as this property follows directly from well-formed infrastructure and that the adding of the token at one end is synchronized to happen after its removal at the other end.

be expressed as $A.\texttt{expectIn}[st]$ `==` `Nil`, where st is the first section of L from A. We can rewrite the above property into the following property. This property corresponds to Step 2 in the verification scheme.

> *"If station A reads from the future for $B.\texttt{rqPerm}$, then at this moment the following holds: $A.\texttt{expectIn}[st]$ `==` `Nil`."*

The formulations differ, as the first condition describes a *state*, but the second one additionally the *history*. We can now formalize the property in ABSDL. This property corresponds to Step 3 in the verification scheme. Step 4 is performed in the proof itself.

Lemma 1. *The following formula holds for all histories generated by the model in Section 4 with a well-formed infrastructure. Let A be a Zmst and L a line bordering A with st being the first section of L from A and $A.\texttt{other}(st)$ the last.*

$$\begin{aligned}&\phi_1(A, st) \equiv\\&\quad \forall i, f.\ h[i] = \mathsf{futREv}(A, \texttt{rqPerm}, f, [\texttt{True}, st]) \rightarrow\\&\qquad\qquad \sigma[i](A) \models \texttt{expectIn}(A.\texttt{other}(st)) = \texttt{Nil}\end{aligned}$$

Proof. To show that claim $\texttt{expectIn}(A.\texttt{other}(st)) = \texttt{Nil}$ holds at the point where `rqPerm` is read it must be shown that `expectIn` is not extended while the process executing `setPreconditions` is suspended. The method that can do so is `offer` . So we have to show that between calling and reading `True` from `rqPerm`, no process that is executing `offer` terminates. We distinguish two cases:

1. `rqPerm` is scheduled after `offer` is called. In this case the station having the token has locked its token—`rqPerm` would return `False`.
2. `rqPerm` is scheduled before `offer` is called. In this case the station requesting the token has locked acceptance—`offer` will not terminate.

The cases correspond to two, intuitively wrong, situations: (1) the token is released by B while it is in the process of dispatching a train (2) a train is accepted by A while it is in the process of requesting the token.

The formal argument is as follows (we mark all properties that were proven mechanically with KeY-ABS with $\mathcal{K}$).[2] First, by the well-formedness axioms there are indices i''', i'', i' with $i' < i'' < i''' < i$ and

$$\begin{aligned}h[i'] &= \mathsf{invEv}(A, B, \texttt{rqPerm}, f, [A, st])\\h[i''] &= \mathsf{invREv}(A, B, \texttt{rqPerm}, f, [A, st])\\h[i'''] &= \mathsf{futEv}(B, \texttt{rqPerm}, f, [\texttt{True}, B.\texttt{other}(st)])\end{aligned}$$

Position i' corresponds to a call on `rqPerm`: the only call is in `setPreconditions` at line 458. We have the following property, because the statement directly before the call has this condition as its guard.

$$\sigma[i'](A) \models \texttt{expectIn}(st) = \texttt{Nil} \wedge \texttt{allowed}[st] = \texttt{False}$$

[2] The model, invariants and KeY-ABS and instructions to compile are available under http://formbar.raillab.de/en/publications-and-tools/latest/

It remains to show that `expectIn` is not modified between the read and the mentioned guard at line 460. The only method adding to `expectIn` is `offer`. I.e, we show that there is no position k with $i' < k < i$ and

$$h[k] = \mathsf{futEv}(B, A, \mathtt{offer}, f', [T, st]))$$

for any `Train` T. Assume there is such a k. Then, by the well-formedness axioms, there are indices k'', k' with $k'' < k'$ and

$$h[k'] = \mathsf{invEv}(B, A, \mathtt{offer}, f', [T, st, B])$$
$$h[k''] = \mathsf{invREv}(B, A, \mathtt{offer}, f', [T, st, B])$$

We have $k' < k'' < k < i$ and make a case distinction

- **Case 1:** $i' < k''$, i.e. the process for `offer` is scheduled after the call on `rqPerm` is made. However, when $A.\mathtt{offer}$ terminates, $A.\mathtt{allowed}[A.\mathtt{other}(st)]$ is set to `True`. This is proven by the following invariant in KeY-ABS:

$$\forall\, \mathtt{Train}\ T.\ \forall\, \mathtt{Section}\ st.\qquad (\mathcal{K})$$
$$\mathit{last}(h) = \mathsf{futEv}(\mathbf{self}, \mathtt{offer}, f, [T, st]) \rightarrow \mathbf{self}.\mathtt{allowed}[st] = \mathtt{True}$$

  Thus $\sigma[k](A) \models \mathtt{allowed}[st] = \mathtt{True}$. But as $\sigma[i'](A) \models \mathtt{allowed}[st] = \mathtt{False}$ holds, it must be set to `True` at some point between i' and k'. The only method setting any key of `allowed` to `True` is `setPreconditions`. Only one such process is active at any one time, thus there cannot be such a modification, and hence no such k' or i'.
- **Case 2:** $k'' < i'$, i.e. the process for `offer` is scheduled before the call on `rqPerm` is made. In this case we cannot rely on the `allowed` field of A. But, $B.\mathtt{unlocked}[st]$ is set to `False` at the moment the call is made, i.e.,

$$\sigma[k'](B) \models \mathtt{unlocked}[st] = \mathtt{False}$$

  This is proven by the following invariant in KeY-ABS:

$$\forall\, \mathtt{Train}\ T.\ \forall\, \mathtt{Section}\ st.\ \forall\, \mathtt{Station}\ B.\qquad (\mathcal{K})$$
$$\mathit{last}(h) = \mathsf{invEv}(\mathbf{self}, B, \mathtt{offer}, f, [T, st, \mathbf{self}]) \rightarrow \mathbf{self}.\mathtt{unlocked}[st] = \mathtt{False}$$

  But when `rqPerm` terminates and returns `True`, then the line must be unlocked. This is proven by the following invariant in KeY-ABS:

$$\forall\, \mathtt{Section}\ st.\qquad (\mathcal{K})$$
$$\mathit{last}(h) = \mathsf{futEv}(\mathbf{self}, \mathtt{rqPerm}, f, [\mathtt{True}, st]) \rightarrow \mathbf{self}.\mathtt{unlocked}[st] = \mathtt{True}$$

  Thus there cannot be such k' or i'. □

5.3 Train Involvement

Railway signals are managed by interlocking systems, but are not detached from the actual movement of the trains: *Zugmitwirkung* ("Train Involvement") is an established concept in German railway operations and states that certain actions

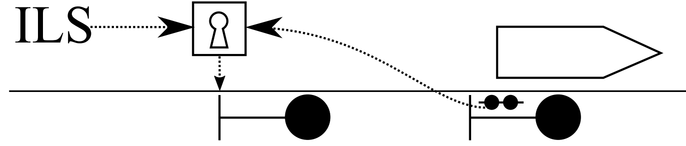

Fig. 5: *Zugmitwirkung* ("Train Involvement"): The train has to trigger the second signal before the first can be set to "Go" again.

of the dispatcher are linked to actions of the train and their detection by the infrastructure. We show the following property, taken from [18]: A signal can only be set to "Go", if the train that passed it the last time has left the covered track. To ensure this, when a signal is set to "Halt", after a train passed it, the used line is *locked*. A signal can no longer be set to "Go" when the route is set to the line while the signal is locked. A signal can only be unlocked when the signal *at the end of the covered section* sends a *backlock* message. Figure 5 illustrates the situation. The desired property, expressed as a statement over states (Step 1 in the verification scheme):

> *"If a non-entry signal S is set to "Go", then the covered section is free of trains going away from it."*

Given the procedure described above, we can again rephrase this into a history-oriented version. For presentation's sake, we do not consider the case that a signal may cover multiple sections. Especially we do not deal with the special requirements for entry signals (Step 2 in the verification scheme).

> *"If a signal S is set to "Go" twice, then a train triggered the point of danger of the next signal at some time in between."*

Using our assumption of well-formed infrastructure (in particular that the `next` field encodes the lines correctly), we can rephrase it more formally with methods and fields as:

> *"If there are two position i, j with $j < i$, such that $h[i]$ and $h[j]$ are invocation reaction events on* `setGo` *on some Signal S covering section S', then there is a k with $j < k < i$ such that $h[k]$ is an invocation reaction event on* `trigger` *on* `next`*(S')."*

We can now formalize the property in ABSDL as an invariant (Step 3 in the verification scheme, Step 4 is again performed in the proof):

Lemma 2. *The following formula holds for all histories generated by the model in Section 4 with a well-formed infrastructure. Let A be a Zmst and S a signal.*

$$\phi_2(A,S) \equiv \forall i.\ \Big(h[i] = \mathsf{invREv}(A, S, \texttt{setGo}, f, []) \rightarrow \\ \forall j.\ \big(j < i \wedge h[j] = \mathsf{invREv}(A, S, \texttt{setGo}, f', []) \rightarrow \\ \exists\ \texttt{DangerPt}\ P.\exists k.\ j < k < i \wedge h[k] = \mathsf{invREv}(P, \texttt{next}(S.covers), \texttt{trigger}, f'', [])\big)\Big)$$

Proof sketch. W.l.o.g we only look at the last such position j. Let S be an exit signal managed by Zfst A and covering section st. Whenever S.`setGo` is called, the managing Zfst has `outLocked`$[st]$ set to `False`. But after a train passed signal S, `outLocked`$[st]$ is set to `True`. In a well-formed infrastructure, before a train passed a signal, no other train is dispatched on the same section, thus `outLocked`$[st]$ must have been set to `False`. The only such method is `backlock` which is only called by the next Zfst once a signal has been triggered. In a well-formed infrastructure, this can only be S.`next`. □

5.4 Discussion

We have shown the following (informal) theorem:

Theorem 1. *Every train departure is safe: when the exit signal S in station A is set to "Go" for train T on a line L to station B, then the first section of L is free, no train is on L in direction of A.*

Formally, this theorem states that the following is an invariant for all histories produced by the model when executed on a well-formed infrastructure.

$$\forall\,\texttt{Zmst}\ A.\ \forall\,\texttt{Section}\ st.\ (\phi_1(A, st) \land \forall\texttt{Signal}\ S.\ \phi_2(A, S))$$

As Lemma 2 also reasons about block signals, we can also state the following:

Corollary 1. *Every train run from station A to station B is safe: during the run, no train will enter the line in the direction of A and whenever a signal is set to "Go", the next section is free.*

Proof sketch. Induction on n, the number of Zfst between A and B.

- Case $n = 0$: In this case this is Theorem 1.
- Case $n = n' + 1$: By induction hypothesis, the train passed the first n' Zfst. By Lemma 1 the permit token cannot be exchanged when the next signal is set to "Go", as the train is still on the line. By Lemma 2 the next section is free, as a Zfst has no entry signals. □

Formally, Corollary 1 states that the following is an invariant for all histories produced by the model when executed on a well-formed infrastructure.

$$\forall\,\texttt{Zmst}\ A.\ \forall\,\texttt{Section}\ st.\ \phi_1(A, st) \land \forall\,\texttt{Zfst}\ Z.\ \forall\texttt{Signal}\ S.\ \phi_2(Z, S)$$

We do not discuss entry signal and entrance into stations. German regulations differ between rules inside and outside of stations and in this work we only reason about the outside rules. The shown properties involve multiple communicating parties. More simple properties can be verified directly by reformulation in ABSDL.

Lemma 3. *If a station A accepts a train t, then there is a track reserved for t. I.e., the following is an ABSDL invariant for the* `ZugMelde` *class:*

$$\forall\,\texttt{Train}\ T.\ \forall\,\texttt{Section}\ st.\ \forall\,\texttt{Station}\ B. \qquad (\mathcal{K})$$
$$\mathit{last}(h) = \mathsf{futEv}(\mathbf{self}, B, \texttt{offer}, f, [T, st, B]) \rightarrow \exists\,\texttt{Signal}\ S.\ \mathbf{self}.\texttt{reserved}[S] = T$$

6 Conclusion

Following the feasibility study [14], the present work is the first time deductive verification is applied to railway operations. Prior verification approaches concentrated on single *components*, not on the *communication structure*, and they mainly used model checking. Our method is not intended to replace model checking of interlocking tables and of consistency properties of the infrastructure. On the contrary, we rely on those results by assuming a *well-formed* infrastructure while reasoning about safety at a higher abstraction level. This allows us to reason globally about systems, however, at the cost of full automation.

Our schema for verifying safety properties with active objects and deductive verification is not limited to the safety properties discussed above. It is as well applicable to other domains than railways, as long as state changes can be associated with events visible in the history. The proofs presented here are a combination of mechanized and pen-and-paper proofs. It would have been possible to formalize and mechanize the whole theory and all proofs. The reasons for the decision to refrain from doing so are twofold: (1) the pen-and-paper approach allows to relate the structure of the proof to informal concepts from the modeled domain, for example, the case distinction in Lemma 1. This strengthens confidence in the model. (2) While the theory of local histories is formalized in KeY-ABS, arguments on the level of multiple objects require a more general logic. KeY-ABS was designed and optimized with the verification of single methods in mind—we conjecture that a formalization of stateful *global* histories is possible, but the required amount of effort does not correlate with the benefits of having more confidence in the proof.

Possibly, an other approach than a purely logical approach to compositional reasoning may be a better fit, however, we do not know of any. The automation of decomposition and localization of safety properties in distributed systems is an open research question.

6.1 Related Work

This is the first full-fledged case study on deductive verification of railway operations, but verification of other aspects of railways has a long tradition which is surveyed in [8].

Verifying railway operations so far has been mostly based on model checking, where the state explosion problem prohibits the reasoning about microscopic models with a large number of participants. To mitigate state explosion, several approaches were proposed. Macedo et al. [16,17] describe a topological decomposition that allows to check the monolithic model of the whole network by checking sub-models. They propose two different cuts to split the monolithic model and corresponding criteria for stations, that ensure that composition is sound. Similarly to our approach, composition guarantees are shown outside of the tool. However, they are still restricted to checking scenarios with a fixed number of model elements, no general infrastructure. Their approach has tool support for OCRA [15,3], a refinement-based approach which models component-based infrastructure with LTL contracts. Cappart et al. [2] verify by simulating the most likely runs in a train station. However, their approach is not exhaustive.

A systematic comparison of the differences between our *modeling* of railway operations and previous approaches to model components can be found in [14].

6.2 Future Work

The split of global invariants into local invariants was performed manually. We plan to formulate safety properties as session types [11], which were recently extended to the ABS concurrency model [13]. This will further automate the verification, while the additional structure of session types allows to relate the strucutre of the proof to real world concepts. We are also interested in the verification of non-functional safety properties, especially deadlock-freedom. We plan to extend the DECO tool [9] with all features needed to handle our model. Furthermore, we plan to model all faults described in [4], fully formalizing and verifying the notion of safety provided there and in related technical documents.

Acknowledgments

We thank the anonymous reviewers for their constructive and valuable feedback. This work is supported by `FormbaR`, "Formalisierung von betrieblichen und anderen Regelwerken", part of AG Signalling/DB RailLab in the Innovation Alliance of Deutsche Bahn AG and TU Darmstadt.

References

1. B. Beckert and R. Hähnle. Reasoning and verification. *IEEE Intelligent Systems*, 29(1):20–29, Jan.–Feb. 2014.
2. Q. Cappart, C. Limbrée, P. Schaus, and A. Legay. Verification by discrete simulation of interlocking systems. In *29th Annual European Simulation and Modelling Conference ESM*, pages 402–409, 2015.
3. A. Cimatti, M. Dorigatti, and S. Tonetta. Ocra: A tool for checking the refinement of temporal contracts. In *28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 702–705, 2013.
4. DB Netz AG, Frankfurt, Germany. Richtlinie 408, Fahrdienstvorschrift, 2017.
5. DB Netz AG, Frankfurt, Germany. Richtlinie 819, LST-Anlagen planen, 2017.
6. C. C. Din, R. Bubel, and R. Hähnle. KeY-ABS: A deductive verification tool for the concurrent modelling language ABS. In A. P. Felty and A. Middeldorp, editors, *CADE*, volume 9195 of *LNCS*, pages 517–526. Springer, 2015.
7. C. C. Din and O. Owe. Compositional reasoning about active objects with shared futures. *Formal Aspects of Computing*, 27(3):551–572, 2015.
8. A. Fantechi, F. Flammini, and S. Gnesi. Formal methods for railway control systems. *STTT*, 16(6):643–646, 2014.
9. A. Flores-Montoya, E. Albert, and S. Genaim. May-happen-in-parallel based deadlock analysis for concurrent objects. In D. Beyer and M. Boreale, editors, *Formal Techniques for Distributed Systems, FMOODS*, pages 273–288, 2013.
10. R. Hähnle. The abstract behavioral specification language: A tutorial introduction. In E. Giachino, R. Hähnle, F. S. de Boer, and M. M. Bonsangue, editors, *Proc. Formal Methods for Component-Based Systems FMCO*, pages 1–37, 2012.
11. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. *SIGPLAN Not.*, 43(1):273–284, Jan. 2008.
12. E. B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, and M. Steffen. ABS: A core language for abstract behavioral specification. In B. K. Aichernig, F. S. de Boer, and M. M. Bonsangue, editors, *Proc. Formal Methods for Components and Objects FMCO*, volume 6957 of *LNCS*. Springer, 2010.

13. E. Kamburjan, C. C. Din, and T.-C. Chen. Session-based compositional analysis for actor-based languages using futures. In *Proc. of the 18th Intl. Conference on Formal Engineering Methods (ICFEM)*, volume 10009 of *LNCS*. Springer, 2016.
14. E. Kamburjan and R. Hähnle. Uniform modeling of railway operations. In C. Artho and P. Ölveczky, editors, *Proc. Fifth Intl. Workshop on Formal Techniques for Safety-Critical Systems (FTSCS)*, volume 694 of *CCIS*. Springer, 2016.
15. C. Limbrée, Q. Cappart, C. Pecheur, and S. Tonetta. Verification of railway interlocking - compositional approach with OCRA. In T. Lecomte, R. Pinger, and A. Romanovsky, editors, *Proc. Reliability, Safety, and Security of Railway Systems. First Intl. Conference, RSSRail*, pages 134–149. Springer, 2016.
16. H. D. Macedo, A. Fantechi, and A. E. Haxthausen. Compositional verification of multi-station interlocking systems. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: 7th Intl. Symp., ISoLA,, Corfu, Greece, Part II*. Springer, 2016.
17. H. D. Macedo, A. Fantechi, and A. E. Haxthausen. Compositional model checking of interlocking systems for lines with multiple stations. In C. Barrett, M. Davies, and T. Kahsai, editors, *NASA Formal Methods: 9th Intl. Symp., NFM, Moffett Field, CA, USA*. Springer, 2017.
18. J. Pachl. *Systemtechnik des Schienenverkehrs: Bahnbetrieb Planen, Steuern und Sichern*. Springer Vieweg, 2008.