Automated Compositional Verification of Interlocking Systems

Haxthausen, Anne E.; Fantechi, Alessandro; Gori, Gloria; Mikkelsen, Óli Kárason; Petersen, Sofie Amalie

Published in:
Proceedings of 5th International Conference Reliability, Safety, and Security of Railway Systems

Link to article, DOI:
10.1007/978-3-031-43366-5_9

Publication date:
2023

Document Version
Peer reviewed version

Automated Compositional Verification of Interlocking Systems

Anne E. Haxthausen[1][0000−0001−7349−8872], Alessandro Fantechi[2][0000−0002−4648−4667], Gloria Gori[2][0000−0002−8482−2612], Óli Kárason Mikkelsen[1], and Sofie-Amalie Petersen[1]

[1] DTU Compute, Technical University of Denmark, Lyngby, Denmark
aeha@dtu.dk
[2] University of Florence, Firenze, Italy
{alessandro.fantechi,gloria.gori}@unifi.it

Abstract. Model checking techniques have often been applied to the verification of railway interlocking systems. However, these techniques may fail to scale to interlockings controlling large railway networks, composed by hundreds of controlled entities, due to the state space explosion problem. We have previously proposed a compositional method to reduce the size of networks to be model checked: the idea is to divide the network of the system to be verified into two sub-networks and then model check the model instances for these sub-networks instead of that for the full network. If given well-formedness conditions are satisfied by the network and the kind of division performed, it is proved that model checking safety properties of all such sub-networks guarantees safety properties of the full network. In this paper we observe that such a network division can be repeated, so that in the end, the full network has been divided into a number of sub-networks of minimal size, each being an instance of one of a limited set of "elementary networks", for which safety proofs have easily been given by model checking once for all. The paper defines a division algorithm, and shows how, applying it to some examples of different complexity, a network can be automatically decomposed into a set of elementary networks, hence proving its safety. The execution time for such a verification turns out to be a very small fraction of the time needed for a model checker to verify safety of the full network.

Keywords: Formal Methods · Model Checking · Compositional Verification · Interlocking Systems.

1 Introduction

Formal methods have successfully been applied to development and verification of railway systems [3, 6, 5]. In particular, model checking techniques have often been applied to the verification of railway interlocking systems. However, model checking is subject to *state space explosion*, which limits scalability of the approach, so that automatic verification of interlocking systems for large networks is demanding in terms of computing resources, and may even fail [4].

Abstraction techniques have typically been adopted to limit state space explosion in model checking. Abstraction should preserve the desired properties and the adopted abstraction technique should be defined specifically for the kind of system and properties under examination. For interlocking systems, a convenient abstraction can be based on the *locality* principle [21, 9]: properties concerning the safe allocation of a route to a train are typically not influenced by other train movements over networks elements that are distant from, and not interfering with, the considered route. Locality of a safety property can be used to limit the state space by abstracting away such "distant movements".

In our previous work, the locality principle is at the base of a *compositional* approach to the verification of interlockings for large networks: the network is divided into two (or more) sub-networks, to which model checking is applied, with a substantial reduction of state space explosion [8, 13, 14, 2]. The soundness result for compositional safety verification given in [8] guarantees that, when properly cutting a network, proving safety for the sub-networks suffices to prove safety for the full network. In this way, the task of proving safety for a large network can be reduced to the task of verifying safety for sub-networks of a size manageable by the model checker.

We have based our compositional approach on the RobustRailS verification framework [20], that exploits the powerful SMT-based RT-Tester bounded model checker[3], although it can be adapted to other verification frameworks: the idea of compositional verification is also shared by the approach described in [10–12]. The two approaches are compared in [1], where it turns out that the latter is grounded on pragmatic domain-related criteria for the definition of how and where to perform the cut into two sub-networks.

The discussion about criteria for localisation of cuts has actually triggered the contribution of this paper, in which a novel iterative decomposition strategy is proposed, to achieve a fine granularity decomposition of a network into a number of small sub-networks, that under certain conditions belong to a library of pre-verified elementary networks. The soundness result for compositional safety verification guarantees that safety for the full network is given by the pre-verified safety of sub-networks. Therefore, to verify a network, it is in principle no more needed to run a model checker, independently of the size of the network, if specific network conditions are met. To the best of our knowledge, we are the first to propose and explore this idea.

The paper formally specifies and implements a division algorithm, and reports on some experiments in which the executable specification as well as the implementation have been applied to some networks of different complexity. In all these experiments, the considered networks were automatically divided into a set of elementary sub-networks, hence proving their safety. In each experiment, the execution time for the algorithm turned out to be a very small fraction of the time needed for a model checker to verify safety of the full network.

After a short description of the RobustRailS verification method in Sect. 2 and a summary of the compositional verification method in Sect. 3, Sect. 4

[3] https://www.verified.de/products/rt-tester/

introduces the possible types of elementary networks, and describes the proposed strategy for performing decomposition into elementary networks. The strategy had been preliminarily sketched in [7]: we now fully formalise it using RSL in Sect. 5, and the executable RSL specification of the division algorithm and its C++ implementation are then applied to some case studies, for which the gains in verification time are shown (Sect. 6). Section 7 draws conclusions and states ideas for future work.

2 The RobustRailS Verification Method and Tools

In the RobustRailS research project[4] that was accompanying the Danish re-signalling programme on a scientific level in 2012–2017, a formal method with tools support for automated, formal verification of railway interlocking systems was developed [20, 18, 19, 17].

About the considered interlocking systems. An *interlocking system* is a signalling system component that is responsible for safe routing of trains through (a fraction of) a railway network under its control.

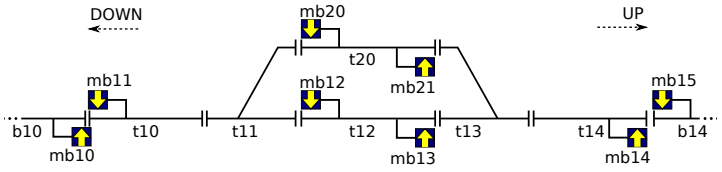


Fig. 1: A railway network layout example. From [18].

In Fig. 1 an example of a railway network layout for a small station is given. As it can be seen, it consists of (1) train detection sections that are either linear sections (like `t10`) or switchable points (like `t11`) having a stem side and two branching sides (e.g. `t11` has its stem next to `t10` and its branches next to `t20` and `t12`, respectively); (2) markerboards[5] (like `mb10`) placed at the ends of linear sections and only visible in one direction (e.g. `mb10` is visible in direction UP). As general rules for the networks considered in this paper, (1) there is at most one markerboard in each end of a linear section, that can only be seen when leaving the section; (2) at the borders of a network, there are always two linear sections (like `b10` and `t10`) with a signal configuration having an *entry signal* on

[4] `http://robustrails.man.dtu.dk`

[5] We are considering modern ERTMS level 2 based interlocking systems for which there are no physical signals. They are replaced by markerboards, and in the control system there are virtual signals associated with the markerboards. Throughout the paper we use the term *signal* as a synonym for *markerboard*.

the border section and an *exit signal* on the section next to the border section. Furthermore, networks are assumed to be *loop-free*[6].

About the tool. The RobustRailS tool, *RR-T*, can be used to verify that an interlocking system instance controlling a certain railway network is safe by giving the tool the following as input: (1) a generic, formal, behavioural model of the interlocking system and generic safety properties, as well as (2) a specification of the network under its control.[7] The tool then checks that the input is wellformed, it instantiates the generic model and generic safety properties with the network description, and finally it verifies that the instantiated model satisfies the instantiated safety properties, by means of a bounded model checker performing a k-induction proof.

3 A Method for Compositional Verification

To introduce the compositional method, we first need to define what is a *cut* of a network, and how the sub-networks should be generated by the cut.

3.1 Cut specifications.

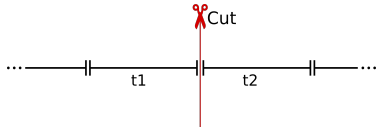


Fig. 2: An example of a single cut. From [8].

A *single cut* is a cut that can be performed between any two neighbouring, non-border sections $t1$ and $t2$ in a network N. An example of a single cut is shown in Fig. 2. The *specification* of that single cut is the pair $(t1, t2)$. To divide a network into two parts, it is not always enough to perform a single cut, but a *cluster cut* consisting of several single cuts may be needed. An example of a cluster cut is shown in Fig. 3. The *specification* of a *cluster cut* is the set of specifications of each of its single cuts. A cut is *legal*, if it divides the network into exactly two parts, no route is cut by more than one single cut, and no flank/front protecting elements[8] are separated by the cut from the sections they protect. In this paper we assume that flank/front protecting is not adopted.

[6] A network is *loop-free*, if there are no physically possible path through the network containing the same section more than once.

[7] Throughout this paper, as generic model and safety properties, we are using those from [20]. The properties are the *no collision* and *no derailment* properties, shared by the vast literature on interlocking verification.

[8] The notion of flank protection is explained in the end of Sect. 4.2.

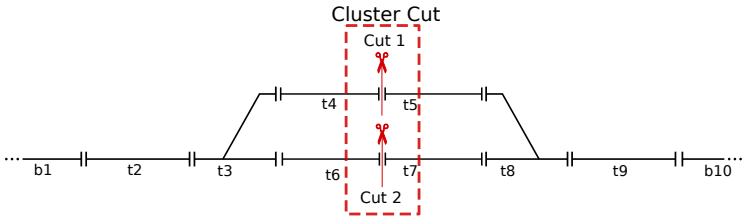


Fig. 3: An example of a cluster cut. From [8].

3.2 Decomposing a network according to a cut specification.

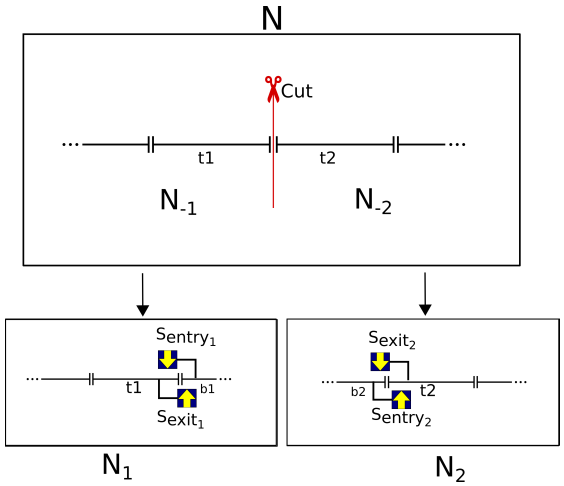


Fig. 4: An example of a decomposition of a network into two networks. From [8].

Given a net N and a legal cut specification, the network can be decomposed into two networks as follows:

- if a single cut is between linear sections $t1$ and $t2$, first divide the network N between $t1$ and $t2$, obtaining two sub-networks N_{-1} and N_{-2}, and then add to N_{-1} and N_{-2} at the respective cut a border section, and also an entry and an exit signal at that border, if there were not already signals placed around the cut. By doing so, two well-formed networks are obtained: N_1 and N_2. Figure 4 shows how a network is decomposed into two networks by a single cut $(t1, t2)$. It can be seen how N_1 is obtained from the sub-network N_{-1} on the left-hand side of the cut by adding a border section $b1$ and border signals s_{entry_1} and s_{exit_1}. N_2 is obtained in a similar way. When it is clear

from the context, sometimes we also call the resulting networks N_1 and N_2 *sub-networks*;
- if a single cut is between a linear section $t1$ and a point p, the decomposition is treated as if there was an additional linear section $t2$ between $t1$ and p, and the cut specification was $(t1, t2)$;
- if a single cut is between two points $p1$ and $p2$, the decomposition is treated as if there were two additional linear sections $t1$ and $t2$ between $p1$ and $p2$, and the cut specification was $(t1, t2)$.
- if the cut is a cluster cut, the above rules are simultaneously applied to each of its single cuts.

3.3 Method steps.

Using a legal cut allows to perform compositional verification in these steps:

1. Decompose a network N according to a legal cut specification, achieving two networks N_1 and N_2.
2. For $i = 1, 2$, apply the RobustRailS tool (RR-T) to N_i to instantiate the chosen generic model and generic safety properties and verify that the instantiated model satisfies the instantiated safety properties.

In [8] it is proved that this method is *sound* and *complete*. Soundness means, that in order to prove safety of the model instance for the whole network, it is sufficient to verify safety for the model instances for the two sub-networks formed by a legal cut. Completeness means, that if the safety proof for one of the sub-networks fails, then one can conclude that safety also fails for the full network.

4 A Decomposition Strategy

Using the presented compositional verification method leaves the question: which cuts should be made in order to decompose a network into small networks that are fast to verify? In this section we will exploit the idea of providing a library of pre-verified, elementary networks and a strategy for dividing a given network into sub-networks of which as many as possible are elementary.

4.1 Elementary Networks

As elementary networks we allow the network patterns shown in Fig. 5: $(a) - (b)$ an *elementary linear network*, that is, a sequence of linear sections having only the required signals at the two borders; $(c) - (d)$ an *elementary point network*, that is, one point surrounded by at least two linear sections on each of its three sides, the required signals at the three borders and optionally zero, one, two or three signals directly facing the point. All patterns admit an unbounded number of linear elements at specific positions. In (c) there is only one linear section

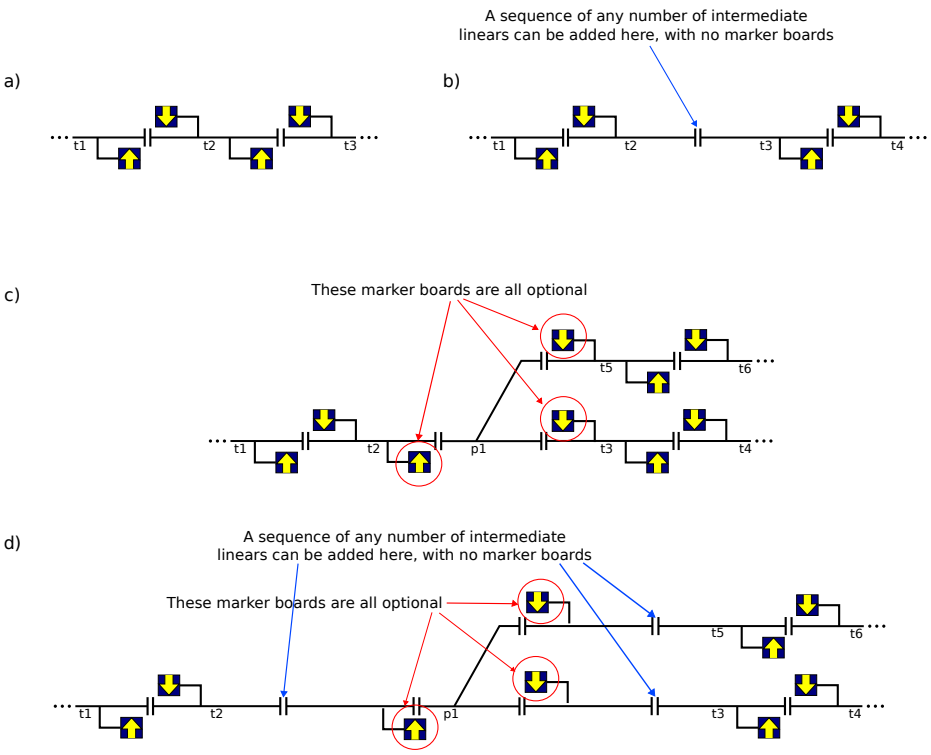


Fig. 5: Patterns for elementary networks.

between the point and each of the three border sections, while in (*d*), there are two (or more) linear sections between the point and each of the three border sections.

Model instances of the networks of Fig. 5 have been model checked to be safe, for all the admitted combinations of presence of markerboards, but without the presence of the admitted extra linear sections. Moreover, a result from [8] allows us to add an unbounded number of linear sections at the indicated specific positions without impacting safety. Hence, we can conclude that *model instances for all elementary networks are safe*.

4.2 Decomposing a network

Given a network, now the idea is to search for places to make legal cuts, one by one, such that the network can be divided into parts that are either *elementary* networks or *non-decomposable* networks (that is, non-elementary networks that cannot be decomposed by any cut(s) without breaking the rules for legal cuts). In the ideal case that the decomposition leads to networks that are all elementary, no additional model checking is needed for verifying the safety of the whole network.

As an example, consider the network shown in Fig. 6. By making the three cuts (two single cuts $(083, PM02U)$ and $(PM02U, PM03U)$ and the cluster cut $\{(802, PM04U), (801, PM04U)\}$) shown by green lines, one by one, one achieves the four elementary networks N_1^1, N_1^2, N_1^3, and N_2^3 shown in Fig. 7.

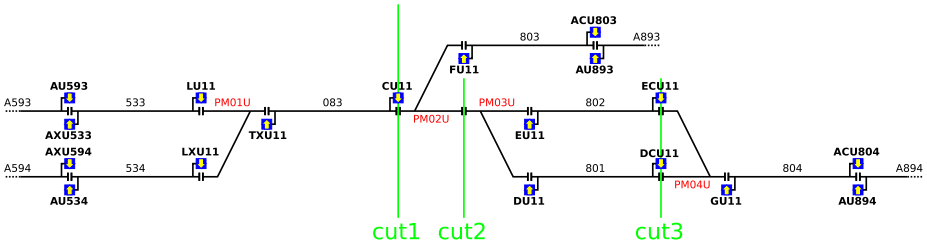
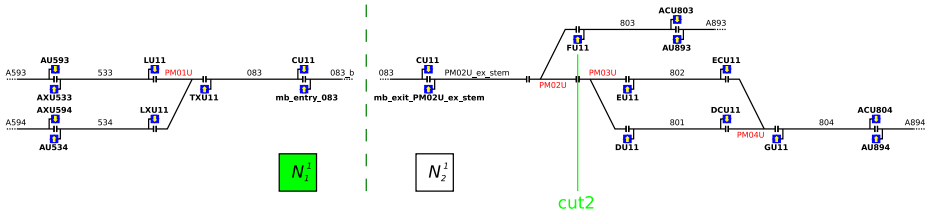


Fig. 6: Cuts shown on a network (LVR1).

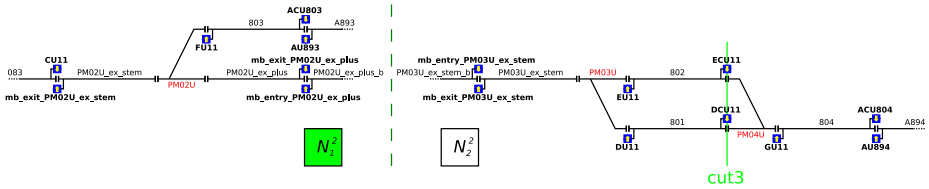
In practice, a possible process of finding such cuts for a loop-free network N is as follows, provided that there are no flank/front protecting elements:

1. Start searching from the neighbour (linear section) l of some border section b of N. The search direction is from l towards the next adjacent element in the direction opposite to b.
2. Follow the sections from l one by one as long as they are linear and do not have any signals attached until one of the following happens:
   (a) If a linear section next to a border is reached, no cut should be made, as the considered network is an elementary linear network.
   (b) If two consecutive, linear sections $l1$ and $l2$ are found, and at least one of them has a signal facing the other, then a decomposition using the cut $(l1, l2)$ should be made. As a consequence, the generated sub-network containing $l1$ will by construction be an elementary linear network. The search for further cuts should then continue from $l2$ in the other sub-network.
   (c) If a point p is found, then we should continue to search for cuts on the two other sides of p. This search depends on from which side p was found: the stem or one of the branching sides. In both cases the search also depends on whether the two other sides are connected or not.[9]
      i. If coming from the stem side of p, and the two other sides are not connected, then we should search for cuts in each of the two other sides. The search here is similar to the search starting from a border, except that if a second point is found, a single cut must be made just *before* that point. The two searches may hence lead to totally zero, one or two single cuts, dividing the network into (1) an elementary

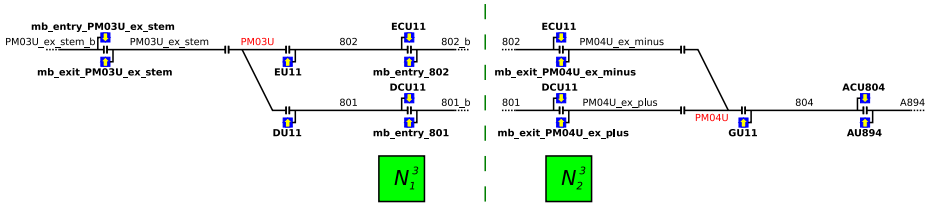
[9] By *connected* we mean that by navigating the graph of the not yet visited part of the network starting from the two sides we eventually reach a common point.



(a) Networks N_1^1+N_2^1 resulting from decomposing the LVR1 network by *cut1*.



(b) Networks N_1^2+N_2^2 resulting from decomposing N_2^1 by *cut2*.



(c) Networks N_1^3+N_2^3 resulting from decomposing N_2^2 by *cut3*.

Fig. 7: Decomposition of the LVR1 network in three steps according to the three cuts shown in Fig. 6. The four resulting green sub-networks N_1^1, N_1^2, N_1^3, and N_2^3 are elementary.

point network containing p and (2) zero, one or two additional sub-networks in which a search for cuts must be performed. For instance, when searching for a cut in network N_2^1 in Fig. 7 (a), starting from $PM02U\_ex\_stem$, a single cut, $cut2 = (PM02U, PM03U)$, will be found in the lower branch, while no cuts are found in the upper branch (as a border is met before any further points or non-border signals), so it results in two sub-networks.

ii. If coming from the stem, and the two branching sides are connected, then a similar search is made in each of the branches. In this case two single cuts (one in each branch) will be found and these must be combined in a cluster cut (in order to divide the network into two parts) leading to an elementary point network containing p and one additional sub-network to which the search for cuts must be recursively applied. That is e.g. the case when searching for a cut (*cut3*) in network N_2^2 in Fig. 7 (b), starting from $PM03U\_ex\_stem$.

iii. If coming from a branching side of p, and the stem and the other branching side are not connected, searches for cuts in the other branch and on the stem side must be performed in a similar way to case i above. That happens e.g. when searching for the first cut in Fig. 6 starting from linear section 533.

iv. If coming from a branching side of p, and the stem and the other branching side are connected, the search to be performed is similar to case ii, except that in some cases it is not possible to find a legal cluster cut: that happens if a potential cluster cut divides a route into three parts[10], as shown in Fig. 8, where the cluster cut shown by a red, dotted line is found when searching from $L1$ on the upper branching side of point $P1$. In such a case we say that N is *unbreakable* from the border b from where the search started, and we should then start a search from another border to see if a cut can be found from there. If N is *unbreakable* from all borders, it is *non-decomposable*. It is our conjecture that it is always possible to find a border from which it is possible to find a legal cluster cut through the connected sub-component, provided that the network is loop-free. For instance, in Fig. 8, the legal cluster cut $\{(P2, P1), (L24, P4)\}$ shown by a dashed, green line can be found when searching from $L2$. Figure 9 gives an example of a network that cannot be decomposed into elementary networks as the network is not loop-free.

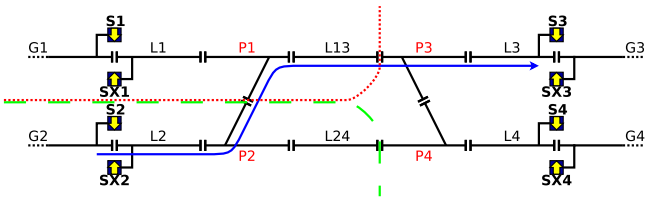


Fig. 8: The cluster cut $\{(P1, P2), (L13, P3)\}$ shown by a red, dotted line is illegal as it divides the route shown as a blue, solid arrow in three parts. The cluster cut $\{(P2, P1), (L24, P4)\}$ shown by a green, dashed line is legal.

In railway interlocking systems, specific additional mechanisms may be included to enforce safety also in the case in which trains do not strictly respect signals, due to a driver's misbehaviour or accidental inability to brake. In the *Flank Protection* mechanism, points and signals not belonging to the route are properly set in order to avoid hostile train movements into the route at an incident point. In the example of Fig. 10 locking of route r requires the point $t20$ to be in the straight position in order to protect the flank of route r by a train

[10] Note that when coming from the stem, we do not have such a problem, as a route cannot pass through a point via its two branches.

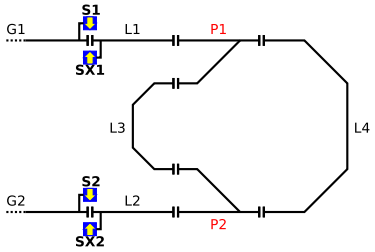


Fig. 9: An example of a non-decomposable network containing a loop.

accidentally missing the closed $mb20$ signal. If both point $t20$ and route r lie in the same sub-network when a cut is operated, the extra condition on the point position has no impact on compositionality: but this is not the case for the drawn cut, which separates the protecting and the protected points. As discussed in [8], in this case compositional verification results do not fully hold, so we consider such a cut as not legal: both elements should instead be in the same sub-network, which is therefore not elementary, since it contains two points. In the presentation of our approach, we have assumed that there is no flank protection. If flank protection was adopted, legal cuts would not be allowed to separate the protecting and the protected points. However, then we would no longer be able to decompose a loop-free network into networks that are all elementary. This also holds for similar protection mechanisms like *front protection* that we do not consider here.

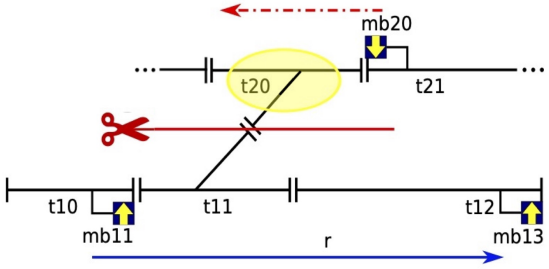


Fig. 10: Cut through a flank protection.

5 Formalisation

The decomposing strategy informally explained in Sect. 4.2 has been formalised as a collection of functions in the formal RAISE Specification Language (RSL) [16]. Below we show sketches of selected parts of that specification.

The specification consists of definitions of the functions and needed data types. There are e.g. data types for representing entities like network layouts and cluster cuts:

```
type
  NetworkLayout = ..., -- network layouts composed of sections and markerboards
  SecId = Text,                -- unique ids for network sections
  MarkerboardId = Text,        -- unique ids for markerboards
  ClusterCut = SingleCut-set,  -- a cluster cut is a set of single cuts
  SingleCut = SecId × SecId,   -- a single cut is a pair of SecIds
  Direction = UP | DOWN, -- two possible search directions
   ...
```

```
decompose : NetworkLayout × SecId × SecId-set →
        NetworkLayout-set × NetworkLayout-set
decompose(N, b, bs) ≡
  let cutset = find_cuts(N,b) in
    if cutset = { {} } -- means N is elementary
      then ({N}, {})
    elsif cutset = {}  -- means N is unbreakable from b
      then -- try to search from another border
        if bs = {} then ({}, {N}) -- no other borders to search from
        else let b2 • b2 ∈ bs in
                decompose(N, b2, bs \ {b2})  -- search from border b2
             end
        end
    elsif card cutset = 1  -- found one non–empty cluster cut
      then
        let
          ccut • ccut ∈ cutset,  -- ccut is the found cluster set
          (N1, N2) = divide(N, ccut) -- divide N into two nets N1 and N2
                                     -- ccut is defined s.t. N1 becomes elementary
          added_borders = ..., -- set of borders added to N2 during division
          b2 • b2 ∈ added_borders,
          (e_ns, u_ns) =
            decompose(N2, b2, (bs ∩ borders(N2)) ∪ added_borders\{b2})
        in
          ({N1} ∪ e_ns, u_ns)
        end
    else -- found two cluster cuts having one single cut each
      -- see text for explanation of what is then done
      ...
    end
  end
```

Fig. 11: Specification of the *decompose* function.

The decomposer has been specified as a recursive function named *decompose* listed in Fig. 11. This function takes as input a network N, a border section b in N (from where a search for cuts should start), and a subset bs of the border sections in N, $b \notin bs$. The set bs is the (current) subset of border sections of N from where there has not yet been made a search. In the first invocation of *decompose*, bs should be the set of all borders, except b. The function returns the set of elementary sub-networks and the set of non-decomposable sub-networks that N can be divided into by following the stepwise process described in Sect. 4.2. Below we will explain the chosen division algorithm for that.

The overall idea of the division algorithm is as follows: in each recursive call of *decompose*, the current network N is divided into two (or three) sub-networks, if possible, of which one is elementary, and then it makes recursive call(s) of *decompose* on the one or two other obtained sub-networks. It will not divide N, if N is already elementary or N is unbreakable from b. In the latter case it will make a recursive call $decompose(N, b2, bs \setminus \{b2\})$, where it starts from another border $b2 \in bs$, if bs is not empty.

The *decompose* function uses an auxiliary function named $find\_cuts$ (listed in Fig. 12 and explained further below) to find the cluster cut(s) that *decompose* should use (in the current iteration) for cutting the network into some sub-networks. Furthermore, it uses another auxiliary function $divide(N, ccut)$ (see [15], where it is named *decompose*) to divide a network N into two sub-networks $N1$ and $N2$, according to a found cluster cut *ccut*. This division is done as informally explained in Sect. 3.2.

When $decompose(N, b, bs)$ is invoked, (in line 4) it invokes $find\_cuts(N, b)$ to find a set *cutset* of next cluster cuts that should be used for cutting the network into sub-networks. Then, depending on the returned set (which by construction will contain zero, one, or two cluster cuts), it will take various actions:
(1) If *cutset* contains one cluster cut which is empty (line 5), it is because N is elementary and the function will (in line 6) return N as the only elementary network and it will return no non-decomposable networks.
(2) If *cutset* is empty (line 7), it is because N is unbreakable from b. In that case, (in line 11) a new search (made by a recursive call of *decompose*) is started from one of the other borders $b2$ of N, from where there has not yet been made a search, and $b2$ is removed from bs. If there was no such other border $b2$, it means that N was unbreakable from any border of N, and the function will (in line 9) return N as the only non-decomposable network and it will return no elementary networks.
(3) If *cutset* contains one non-empty cluster cut *ccut*, it will use $divide(N, ccut)$ (in line 18) to divide N into two networks $N1$ and $N2$, where $N1$ will be elementary due to the definition of $findcuts$ and *divide*. Then, (in line 23), it will continue making a recursive call of *decompose* on $N2$ from one of $N2$'s added borders $b2$, obtaining a set of elementary networks e_ns and set of non-decomposable networks u_ns. Finally (in line 25), it will add $N1$ to e_ns.
(4) In a similar way, if *cutset* contains two cluster cuts (line 26), it will make two consecutive calls of *divide* using these cluster cuts to divide N into three

networks $N1$, $N2$, and $N3$, of which $N1$ will, by construction, be elementary. Then it will make two recursive calls of *decompose* on $N2$ and $N3$, respectively, obtaining two sets of elementary networks, e_ns2 and e_ns3, and two sets of non-decomposable networks, u_ns2 and u_ns3. Finally, it will return $(\{N1\} \cup e\_ns2 \cup e\_ns3, u\_ns2 \cup u\_ns3)$.

The termination of *decompose* is guaranteed by the fact that each time it is invoked in cases (3) and (4), it is called with smaller networks, and in case (2) the bs parameter becomes smaller.

```
1  find_cuts : NetworkLayout × SecId → ClusterCut-set
2  find_cuts(N, b) ≡
3    let
4      dir = find_direction_towards_neighbor_of_border(b, N),
5      l = next_from_linear(b, dir, N) -- l is the neighbor of b
6    in
7      find_cuts_from_linear(l, dir, N)
8    end,
9
10 find_cuts_from_linear : SecId × Direction × NetworkLayout → ClusterCut-set
11 find_cuts_from_linear(l, dir, N) ≡
12   let next = next_from_linear(l, dir, N) in  -- next section to visit
13     if is_linear(next, N)
14     then
15       if is_border(next, N) then {{}}   -- case 2(a)
16       elsif has_signal(l, dir, N) ∨ has_signal(next, opposite_direction(dir), N)
17           then { { (l , next) } }          -- case 2(b)
18       else -- no signals between l and next
19         find_cuts_from_linear(next, dir, N) -- continue search from next
20       end
21     else -- is_point(next, N), i.e. case 2(c)
22       -- further search depends on from which side the point is met:
23       case get_pointend_entry_given_neighbor(next, l, N) of
24         STEM → find_cuts_from_stem(next, dir, N),
25         PLUS → find_cuts_from_branch(next, dir, PLUS, N),
26         MINUS → find_cuts_from_branch(next, dir, MINUS, N)
27       end
28     end -- if
29   end -- let
```

Fig. 12: Specification of the $find\_cuts$ function.

$find\_cuts(N, b)$ (listed in Fig. 12) takes as input a network N and a border section b of N (from where the search should start) and searches for cuts that can be used to divide the network such that one of the resulting sub-networks is an elementary network containing b. This search is done by invoking (in line 7) another auxiliary function $find\_cuts\_from\_linear(l, dir, N)$ (also listed in

Fig. 12) to search for cuts from the linear section l next to b in the search direction *dir* going from b towards l. This formalises step 1 in Sect. 4.2.

$find\_cuts\_from\_linear(l, dir, N)$ (listed in Fig. 12) takes as input a network N, a linear section l of N (from where the search should start), and a search direction *dir*. It returns a set of "next" cluster cuts that can be found when searching from l in direction *dir*. This set will by construction contain zero, one, or two cluster cuts. The search is made as described under step 2 in Sect. 4.2, and the returned set of cluster cuts, will contain the cuts found as explained informally for each of the cases 2(a), 2(b), 2(c)i - 2(c)iv in Sect. 4.2.

The function uses two auxiliary functions, $find\_cuts\_from\_stem$ and $find\_cuts\_from\_branch$, to specify the search for case 2(c). They are defined in a similar way as $find\_cuts\_from\_linear$, but not shown here due to space limitations.

6 Experiments

The formal RSL specification of the *decompose* function is executable and has hence been used as an early prototype for a decomposer tool. It has been thoroughly tested to be functionally correct. After that the specification was translated to C++ achieving a second prototype which was also tested. The tests have shown a full agreement between the two prototypes, and have therefore given some confidence in the correctness of the algorithm and of its implementation, although we have not attempted a formal proof thereof.

Furthermore, we have used first the RSL executable and later the C++ executable for making some experiments: for networks of various complexity, we measured the time it takes to automatically decompose a network into elementary networks and we compared this with the time it takes to verify the full network using the RobustRails Tool (RR-T).

Table 1: Verification metrics for the RobustRails Tool (RR-T) and the decomposer prototypes (in RSL and C++) applied to some interlocking examples. Time is measured in seconds.

Example	Linears	Points	Signals	Routes	RR-T Time	RSL Time	C++ Time	Sub-networks
EDL	111	39	126	179	22863	219	1,5	68
LVR1	11	4	18	18	91	7	0,5	4
LVR7	26	12	42	48	49813	9	0,5	13
Tramway line	22	12	20	62	43184	8	0,5	12
Flying junction	24	16	16	40	62172	9	0,7	16

Table 1 shows the metrics for these experiments using the RobustRails Tool (RR-T) and the decomposer prototypes (both the RSL specification and the C++ implementation). The tools have been applied on a benchmark of network layouts considered in some other, past experiments (using the RobustRailS Tool only), some of which were published in previous conference papers [1, 2, 14, 12].

The EDL network is a line in Denmark comprising 8 stations, LVR1 is Binche station in Belgium, LVR7 is Piéton station in Belgium, and the two last networks are inspired by real networks [7]. In the table, columns 2-5 give for each network example its number of linear sections, points, signals, and routes, respectively. Column 6 shows for each network the verification time using the RobustRails Tool and columns 7 and 8 show the average time[11] needed to divide the network into sub-networks using the decomposer prototypes. The last column contains the number of elementary sub-networks obtained. In all cases no non-decomposable sub-networks were obtained. All the experiments were executed on an Intel Core i5 CPU 750 (-MPC-) at 1.20GHz, 16GB RAM, Ubuntu 14.04, Linux 3.19.0-25-generic x86 64 (64 bit, gcc: 4.8.2) kernel.

The experiments show a dramatic reduction of the time needed for the verification of a network using our decomposer prototypes compared to time needed when using the traditional monolithic verification by the RobustRails Tool. The considered networks have all been successfully decomposed into elementary networks, in most cases in as many as there were points in the network, that is, in elementary point networks of type (c) or (d) in Fig. 5.

7 Conclusions and Future Work

In this paper, we have exploited a previously defined compositional method for model checking the safety of interlocking systems, by pushing it to the finest granularity level. The said compositional method guarantees that, under given conditions, dividing into two parts a network expressing the interlocking over a complex network layout and then proving safety of the two parts equates to proving safety of the whole network. This provides significant advantages in terms of reduction of state space explosion.

In this paper, we have formally specified in RSL and implemented in C++ an algorithm which automatically divides a network into a number of sub-networks of minimal size by repeatedly applying the above mentioned network division. In the ideal case each of the resulting sub-networks is an instance of one of a limited set of "elementary networks", for which safety proofs have already been given (in less than 3 seconds) by model checking, once for all. That means, in such a case no model checking is needed. We have successfully applied first the fully automated RSL executable and later also the C++ implementation to decompose into elementary networks several network examples of different complexity, hence proving their safety. In all cases the execution time was a very small fraction of the time needed by a model checker to verify the full network.

[11] The *decompose* function was repeatedly invoked with each of the network's borders as the border parameter b of *decompose*, and then the average execution time was computed. Note that the invocations with different b on the same network sometimes produce slightly different sets of networks: the cardinalities of the sets of networks returned by two different invocations are the same and networks are usually the same, except that sometimes a linear section may be included in a different sub-network.

Our suggested verification approach will be the subject of further work: For all the case studies, the algorithm succeeded to divide the network into sub-networks that were all elementary. In principle, the algorithm could return one or more sub-networks that cannot be decomposed into elementary networks, but we have not found any loop-free networks for which this is the case, provided that there is no flank/front protecting elements. We conjecture that the algorithm is always capable to divide any loop-free network into elementary networks, provided that there is no flank/front protecting elements, but we need to formally prove this conjecture.

A proof of correctness of the algorithm could be a topic for future work: if the conjecture above is proved to hold, the algorithm should be demonstrated to produce a consistent set of elementary networks.

For future work, it could also be interesting to investigate how the topology and the choice of the border from where the search should start impact the execution time of the decomposer prototypes.

We conjecture that the proposed decomposition method can be instantiated with similar benefits in other compositional frameworks, as the one described in [1]: this is another future research direction.

Acknowledgements

The authors would like to thank (1) Jan Peleska and Linh H. Vu together with whom Anne Haxthausen developed the RobustRailS verification method and tools, (2) Hugo D. Macedo, who contributed to the initial work on the applied compositional method, and to (3) Anna Nam Anh Nguyen and Ole Eilgaard for their network cutter tool which we have integrated in our decomposer tool.

The contribution by the second and third author was carried out within the MOST – Sustainable Mobility National Research Center and received funding from the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.4 – D.D. 1033 17/06/2022, CN00000023). This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

References

1. Fantechi, A., Gori, G., Haxthausen, A.E., Limbrée, C.: Compositional verification of railway interlockings: comparison of two methods. In: Dutilleul, S.C., Haxthausen, A.E., Lecomte, T. (eds.) Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification: Fifth International Conference, RSSRail 2022, Paris, France, June 1-2, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13294, pp. 3–19. Springer Nature Switzerland AG (2022)
2. Fantechi, A., Haxthausen, A.E., Macedo, H.D.: Compositional verification of interlocking systems for large stations. In: Cimatti, A., Sirjani, M. (eds.) Software Engineering and Formal Methods - 15th International Conference on Software Engineering and Formal Methods, Trento, Italy, September 4-8, 2017. Lecture Notes in Computer Science, vol. 10469, pp. 236–252. Springer (2017)

3. Ferrari, A., ter Beek, M.H.: Formal methods in railways: A systematic mapping study. ACM Comput. Surv. **55**(4), 1–37 (nov 2022)
4. Ferrari, A., Magnani, G., Grasso, D., Fantechi, A.: Model Checking Interlocking Control Tables. In: FORMS/FORMAT 2010 – Formal Methods for Automation and Safety in Railway and Automotive Systems. pp. 107–115. Springer (2010)
5. Ferrari, A., Mazzanti, F., Basile, D., ter Beek, M.H.: Systematic evaluation and usability analysis of formal methods tools for railway signaling system design. IEEE Transactions on Software Engineering **48**(11), 4675–4691 (2022)
6. Ferrari, A., Mazzanti, F., Basile, D., ter Beek, M.H., Fantechi, A.: Comparing formal tools for system design: A judgment study. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. pp. 62–74. ICSE '20, Association for Computing Machinery, New York, NY, USA (2020)
7. Haxthausen, A.E., Fantechi, A., Gori, G.: Decomposing the verification of interlocking system. In: Haxthausen, A.E., Huang, W., Roggenbach, M. (eds.) Applicable Formal Methods for Safe Industrial Products - Essays Dedicated to Jan Peleska on the Occasion of His 65th Birthday. Lecture Notes in Computer Science, vol. 14165 (Accepted for publication in 2023)
8. Haxthausen, A.E., Fantechi, A.: Compositional verification of railway interlocking systems. Form. Asp. Comput. **35**(1) (2023). https://doi.org/10.1145/3549736
9. James, P., Möller, F., Nguyen, H.N., Roggenbach, M., Schneider, S., Treharne, H.: Decomposing scheme plans to manage verification complexity. In: Schnieder, E., Tarnai, G. (eds.) FORMS/FORMAT 2014 - 10th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems. pp. 210–220. Institute for Traffic Safety and Automation Engineering, Technische Univ. Braunschweig (2014)
10. Limbrée, C., Cappart, Q., Pecheur, C., Tonetta, S.: Verification of Railway Interlocking - Compositional Approach with OCRA. In: Lecomte, T., Pinger, R., Romanovsky, A. (eds.) Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification: Second International Conference, RSSRail 2016, Paris, France, June 28-30, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9707, pp. 134–149. Springer Cham. (2016)
11. Limbrée, C., Pecheur, C.: A Framework for the Formal Verification of Networks of Railway Interlockings - Application to the Belgian Railway. Electronic Communication of the European Association for the Study Science and Technology **76** (2018)
12. Limbrée, C.: Formal verification of railway interlocking systems. Ph.D. thesis, UCL Louvain (2019)
13. Macedo, H.D., Fantechi, A., Haxthausen, A.E.: Compositional verification of multistation interlocking systems. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation, Part II. Lecture Notes in Computer Science, vol. 9953, pp. 279–293. Springer International Publishing AG (2016)
14. Macedo, H.D., Fantechi, A., Haxthausen, A.E.: Compositional Model Checking of Interlocking Systems for Lines with Multiple Stations. In: Barrett, C., Davies, M., Kahsai, T. (eds.) NASA Formal Methods: 9th International Symposium, NFM 2017, Proceedings. pp. 146–162. Springer International Publishing (2017)
15. Nguyen, A.N.A., Eilgaard, O.B.: Development and use of a tool supporting compositional verification of railway interlocking systems. Tech. rep., DTU Compute, Technical University of Denmark (2020), `https://findit.dtu.dk/en/catalog/5f181f35d9001d016b4e1f3b`

16. The RAISE Language Group: Chris George, Peter Haff, Klaus Havelund, Anne E. Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, Kim Ritter Wagner: The RAISE Specification Language. The BCS Practitioners Series, Prentice Hall Int. (1992)
17. Vu, L.H., Haxthausen, A.E., Peleska, J.: A Domain-Specific Language for Railway Interlocking Systems. In: Schnieder, E., Tarnai, G. (eds.) FORMS/FORMAT 2014 - 10th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems. pp. 200–209. Institute for Traffic Safety and Automation Engineering, Technische Universität Braunschweig (2014)
18. Vu, L.H., Haxthausen, A.E., Peleska, J.: A domain-specific language for generic interlocking models and their properties. In: Fantechi, A., Lecomte, T., Romanovsky, A. (eds.) Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification: Second International Conference, RSSRail 2017, Pistoia, Italy, November 14-16, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10598, pp. 99–115. Springer Cham (2017)
19. Vu, L.H.: Formal Development and Verification of Railway Control Systems - In the context of ERTMS/ETCS Level 2. Ph.D. thesis, Technical University of Denmark, DTU Compute (2015)
20. Vu, L.H., Haxthausen, A.E., Peleska, J.: Formal modelling and verification of interlocking systems featuring sequential release. Science of Computer Programming **133, Part 2**, 91–115 (2017)
21. Winter, K.: Optimising Ordering Strategies for Symbolic Model Checking of Railway Interlockings. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies. Lecture Notes in Computer Science, vol. 7610, pp. 246–260. Springer Berlin Heidelberg (2012)