ELSEVIER

Mathematics and Computers in Simulation 45 (1998) 175–183

MATHEMATICS AND COMPUTERS IN SIMULATION

# An applicable topology-independent model for railway interlocking systems

Eugenio Roanes-Lozano[a,*], Luis M. Laita[1,b]

[a] *Dept. Algebra, Universidad Complutense de Madrid, Madrid, Spain*
[b] *Dept. Artificial Intelligence, Universidad Politécnica de Madrid, Madrid, Spain*

**Abstract**

In this article, two decision models for railway interlocking systems are presented. The first – for the case that the turnouts have spring switches- and the second – for the case that the turnouts should not be trailed through a switch set against-. In both cases, the algorithm is independent of the topology of the station and is based on the calculation of accessibility in an oriented graph.

Both models are surprisingly brief and easy to implement in Maple V.4. The package analyzes any change in the position of the switches and the clearances given by the semaphores, and supervises if it is safe to authorize the change. Any turnout, that could be trailed through a switch set against, is also studied in the second model.

In both cases, the fact that trains could occupy more than one section is considered. Some examples are included. © 1998 IMACS/Elsevier Science B.V.

## 1. Introduction

### *1.1. Some vocabulary*

As a guided transportation system, a train can move from one track to another only at certain places, where an adequate device is installed (turnout). The turnout has a mobile part (switch), that sends the trains in one of two directions (direct track/diverted track).

Let us suppose that a certain turnout connects section $A$ with sections $B$ and $C$ ($B$ on the straight route). If trains coming from $A$ are sent to $B$ (straight route), there would be no problem for a train coming from $B$, as it would pass to section $A$. But if the train were coming from $C$, it would trail through a switch set against it. In such a case there would be two possibilities:

* Corresponding author. E-mail: eroanes@eucmos.sim.ucm.es
[1] E-mail: laita@fi.upm.es.


*PII* S0378-4754(97)00093-1

- The turnout has a modern spring switch: nothing happens, the train passes to section $A$ and, afterwards, the spring returns the switch to the original position (this will be the case studied in Model 1). There is also no problem if the switch is an old-style tramway stub switch.
- The turnout has a normal switch: the train cannot pass to section $A$. Normally, the switch is damaged and the train can derail (this will be the case studied in Model 2).

### *1.2. The decision problem*

Let us suppose that the stationmaster at a certain station gives clearance to a train. This action should not lead to any collision, nor to any conflict between the setting of signals and switches. When there are several trains, signals and turnouts involved, it is not a trivial problem to prevent collisions.

From the early times of railway security systems, several approaches have been used. In the beginning, complicated mechanical interlockings (that used bars) were developed, not to allow some immediately conflicting actions (for instance, to give clearance to two trains meeting at the next turnout or to set to green two semaphores; one that allows leaving the station and another that allows entering it – at the same side of the station-). The first such device seems to have been installed by Saxby in 1859, near London [7].

From the second quarter of this century, relays have been used for the same purpose. Obviously, the installations could only be topology-dependent, and very complicated to design.

In the eighties, high-tech companies, such as Siemens, began to install microcomputer-controlled interlocking systems ([8–11]).

Technology in railways is very conservative. As it happens in aeroplane technology, until a technology is really thoroughly tested, it is not trusted. That is the reason for railway technology to seem, sometimes, obsolete. For instance, interlocking using relays are still being installed in 1996.

We shall deal only with the decision problem (the ''Logical'' problem). There are some other topics (redundancy of equipment, interfaces computer↔railway hardware,. . .) that will not be treated here.

### *1.3. Our concept of complete safety*

A quasi-inductive definition of ''safe position'' will be used. Any situation will be safe, if the first situation ($P_1$) was safe and, if the actual situation is ($P_i$), a new situation ($P_{i+1}$) is to be analyzed to be safe before being authorized.

In the course of this article, the fact that a train can move forward and backwards as many times as desired (obeying the signals) is considered.

## 2. The approach

Initially, a classical bivalued Logic rule based KBS was intended to be adapted (an approach similar to that of Refs. [1–4]). But, attacking the problem, as in Graph Theory, led to a very clear approach, as presented here.

### *2.1. Accessibility to the next sections*

We consider two oriented graphs. In one of them (corresponding to turnouts), there is an edge connecting section $A$ and section $B$ if, and only if, there is a turnout connecting sections $A$ and $B$ and

the switch is in the right position. The associated Boolean matrix (with 0s and 1s) will be denoted by *MD*.

In the other oriented graph (corresponding to semaphores), there is an edge connecting section *A* with section *B*, unless there is a semaphore controlling the pass from section *A* to section *B* and it is red. The associated Boolean matrix will be denoted by *MS*.

In this way, a next section is accessible from another if, in both directed graphs, there are edges connecting them. The matrix associated with the merged graph can be obtained on multiplying element by element *MD* and *MS* (this multiplication is not the usual matrix product: it is the operation corresponding to the logical connective ''and'').

### *2.2. General accessibility*

The problem is slightly more complicated: as a train can move from one section to the next, and then to the next to this second one,... The solution is to calculate the transitive closure of the merged graph.

It can be calculated by multiplying the matrix of the merged graph by itself until it stabilizes. The associated matrix is denoted by *MA*.

## 3. Algorithm – simplified case-

### *3.1. Looking for possible collisions – simplified case- (Models 1 and 2)*

Each train is supposed to occupy a single section ([5,6]).

The position of the trains is given as a list of sections. Each number is then translated into a row matrix of 0s with a 1 in the corresponding position. The sections accessible from a section can be obtained by multiplying the correspondent row matrix by *MA* (the resultant row matrix will be denoted as ''access row matrix'').

If this is done for each train, no section should be accessible by more than one train. This can be checked by adding the accessible row matrices and looking for values >1 in the resulting row matrix.

### *3.2. Looking for possible trailing through switches set against (Model 2)*

One more matrix, similar to *MD*, is considered. The difference is that it has 0s, 1s and 2s. When moving from one section to another is not allowed there is a 0, when it is possible there is a 1, and when there is a trailing through a switch set against, there is a 2.

It only has to be checked if, from any of the accessible sections for any of the trains, a 2 can be obtained.

## 4. Algorithm – general case-

### *4.1. Improvements w.r.t. the simplified case*

In previous versions of this work ([5,6]), trains were supposed to occupy a single section (this is a shortcoming, since this could be false, as for instance for a long goods train).

The shortcoming has been solved in this new version. But some changes in the algorithms had to be made.

### *4.2. Changes w.r.t. the simplified case*

In this general model, the position of the trains is given, not as a list of sections, but as a list $L$ of lists of sections. Each of these lists (members of the given list $L$) are the sections occupied by one single train. The sections occupied by a train are supposed to be given in the correct order (locomotive→end, or vice versa).

Including the fact that trains can occupy more than a single section has two effects on the algorithm:

(i) The accessibility from one train is not just the accessibility from a single section, but from several ones.
(ii) It has to be checked that no switch under any train has been changed.

The solution to (i) is simple but tricky. The engine driver of the train can be in the locomotive or in the last coach or wagon (caboose). Therefore, only the accessibility from the two ends of the list has to be checked and merged. But these two row matrices should not be added as before because, now, we are not dealing with accessibility of two trains, but accessibility from the two ends of the same train.

Surprisingly, this is not enough to completely describe the accessibility, because the intermediate sections are also occupied (even if the train cannot move!). Hence, a row matrix with the sections occupied by the train has to be merged with the two previous ones. To do so, a procedure that calculates the matrix of the maximums of three given matrices (element by element) is defined.

The solution to (ii) is not as simple as it seems to be. Initially, we thought about checking if all its intermediate sections could be reached from both ends of the train. But, we found wrong answers. The reason is that this could be done not by moving to the adjacent section but through a complicated itinerary.

Therefore, what is checked is: if the adjacent sections of the train can be reached from any given section. To do so, the matrix that has to be used is not the matrix of transitive closure of the merged graph of semaphores and turnouts. It is just the matrix of turnouts (without considering its transitive closure!). This ''accessibility2'' is calculated by means of a couple of new procedures.

Looking for possible trailing through switches set against in the model 2 is almost unaffected by these changes (only minor changes had to be made, as the input is a list of lists instead of a list).

## 5. Use of the package

(Hereafter, all the arguments refer to the ''general case'').

### *5.1. Using Model 1*

Although the user interface is not developed, it is very easy to use. Once Maple V.4 is started, the `linalg` package has to be loaded (to perform matrix calculations)

```
with(linalg);
```

and then the interlocking package

```
read('enclav1n.map');
```

Now the data of the station has to be introduced. With `initial(n);` the number of sections is given. The procedure `turnout(a,b,c,pos)` is used to introduce the position of the switches, as follows:

- To type `turnout(a,b,c,pos);` means that there is a turnout connecting section `a` to both `b` – through a straight route – and `c`, and now the switch is in straight route (trains will run on the direct track).
- To type `turnout(a,b,c,1);` means that there is a turnout connecting section `a` to both `b` – through straight route – and `c`, and now the switch is not in straight route (trains will run on the diverted track).

The procedure `semaphore(a,b,pos)` is used to introduce the colour of the semaphore controlling the pass from section `a` to section `b` as red (`semaphore(a, b,0)`) or green (`semaphore(a,b,1)`).

Then it is possible to look for the possible collisions just by typing

```
look_for_crash([[n11,n12,...],[n21,n22,...],...]);
```

(where `[n11,n21,...]` is the well-ordered list of sections occupied by a certain train, `[n21,n22,...]` is the well-ordered list of sections occupied by another train,...).

If a problem is detected, a help to avoid it is to ask for the sections reachable from a given section with

```
accessible(n);
```

Also, it has to be checked if all the switches under every train are positioned correctly. This is done by

```
look_under();
```

(that should be executed only after having executed `look_for_crash([[...],[...],...]);`).

Note that, if any change is introduced in the position of the trains, it is necessary to run `look_for_crash([[...],[...],...]);` for the new situation, before running again `look_for_against();` (anyway, if any change takes place it is strongly recommended to run `look_for_crash([[...],[...],...]);`).

### *5.2. Using Model 2*

The only difference lies in that the second part of the package also has to be loaded. Thus, in the beginning, it should be typed

```
with(linalg);
read('enclav1n.map');
read('enclav2n.map');
```

After the data of the station is introduced, and after `look_for_crash` has been executed, it can also be asked if any trailing through switches set against could take place, by typing

```
look_for_against();
```

Note that, if any change is introduced in the position of the switches, colour of the semaphores or position of the trains, it is necessary to run `look_for_crash([[...],[...],...]);` for the new situation, before running `look_for_against();` again.

## 6. A very simple example (Model 1)

The situation in Fig. 1 (trains are supposed to be driven on the right-hand track and, therefore, signalling is always on the right side the track) can be introduced by typing

```
initial(6);
turnout(1,3,2,0);
turnout(2,5,4,1);
turnout(6,3,4,0);
semaphore(3,1,0);
semaphore(4,2,0);
```

If a train ($\alpha$) is occupying sections 1, 2, 4 and another train ($\beta$) is in section 6, the situation can be checked with

```
look_for_crash([[1,2,4],[6]]);
```

that answers

```
There could be a collision in section 3
There could be a collision in section 6
```

Observe that the train $\alpha$ could move backwards to section 1 (trailing through a switch set against is allowed in this model). Then it could move towards sections 3 and 6, where it could collide with train $\beta$ (in section 3 or 6, depending on whether $\beta$ had stayed in 6 or had moved to 3).

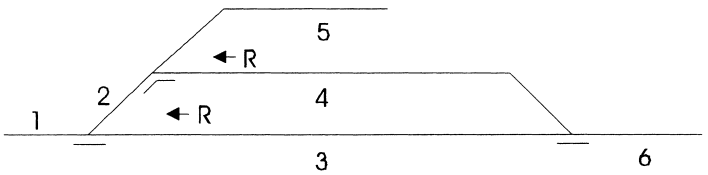


Fig. 1.

The case that train $\alpha$ could move from section 4 directly towards section 6 and collide with train $\beta$ in section 6 is also computed (that movement would produce a derailment or a brake of the couplings or the switch (1,3,2) – because of the position of the switch of the turnout (1,3,2-).

This problem is detected by

```
look_under();
```

that answers

```
There is an incorrectly set switch under train [1,2,4]
```

If the switch (1,3,2) is changed and the semaphores controlling the movement from sections 1 to 3 and 4 to 6 are set red

```
turnout(1,3,2,1);
semaphore(1,3,0);
semaphore(4,6,0);
```

then

```
look_for_crash([[1,2,4],[6]]);
look_under();
```

indicate no problem.

## 7. A not-so-simple example (Model 2)

The situation in Figs. 2 (signalling is on the right side of the track) can be introduced by typing

```
initial(23);
turnout(1,2,11,0);
```

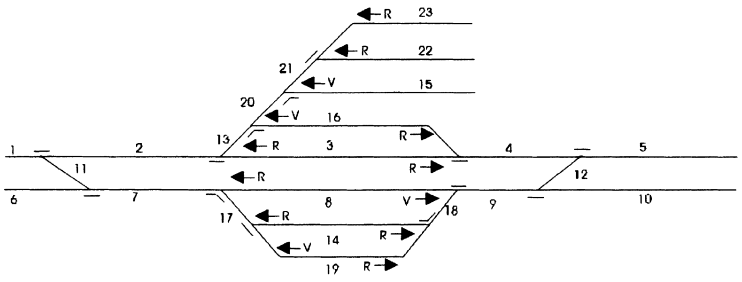


Fig. 2.

```
turnout(7,6,11,0);
turnout(2,3,13,0);
turnout(13,20,16,1);
...
semaphore(16,13,1);
semaphore(3,2,0);
...
```

Let us suppose that there are six trains situated, respectively, at sections 1, 3, 15, 9, 14 and 23. Let us check their situation

```
look_for_crash([[1],[3],[15],[9],[14],[23]]);
```

answers

```
There could be a collision in section 3
```

and

```
look_for_against();
```

answers

```
Trailing through switch set against could take place from section 20 to 13
```

Meanwhile, `look_under();` finds no problem (what is trivial, as all the trains occupy, in this case, one section).

Let us now consider five trains positioned as follows: one in sections 3,4,5, another one in sections 7 and 8 and the other three, respectively, in sections 15, 22 and 23. Now

```
look_for_crash([[3,4,5],[7,8],[15],[22],[23]]);
```

finds no problem. But

```
look_for_against();
```

finds the same problem as before

```
Trailing through switch set against could take place from section 20 to 13
```

And, in this case, `look_under();` answers

```
There is an incorrectly set switch under train [7,8]
```

(if it moves in one direction it will probably derail, and in the opposite one it will trail through a switch set against; the problem is with `turnout(7,8,17)`).

## 8. Remarks and conclusions

We think this work presents a fully automated and original approach to a non-trivial decision problem.

Moreover, the simplicity of the algorithms and procedures is remarkable (the Maple files `enclav1n.map` and `enclav2n.map` occupy, respectively, 3.5 and 1.5 pages – including comments – in print).

This work is a further development of the theory presented in Refs. [5,6] (see Section 4).

Matrices are very sparse and Boolean in the first model (or with 0s, 1s and 2s in the second one). Therefore, a special treatment for sparse matrices could be used.

The development of an implementation directly in *C* is being considered now (for the sake of speed and compiling capability).

The use of a computer algebra system is not strictly necessary as the distinctive abilities of these systems are not used (exact arithmetic and non-assigned variable handling). Nevertheless, the favourable environment accompanying to a CAS (Maple V.4 in this case), made the implementation much more simple and straightforward than using a classic language.

## Acknowledgements

This work was partially supported by DGICYT of Spain, PB 94-0424.

## References

[1] L.M. Laita, L. de Ledesma, E. Roanes-L., E. Roanes-M., An Interpretation of the Propositional Boolean Algebra as a *k*-algebra. Effective Calculus. LNCS 958, Springer-Verlag, 1995, (pp. 255–263).
[2] E. Roanes-L., L.M. Laita, E. Roanes M: Maple V in A.I.: The Boolean Algebra Associated to a KBS. CAN Nieuwsbrief 14, 1995, 65–70.
[3] E. Roanes-L., L.M. Laita, Verification of Knowledge Based Systems with Commutative Algebra and Computer Algebra Techniques, in: Proceedings of the 1st International Conference on Applications of Computer Algebra (IMACS), New Mexico University, 1995, electronic book.
[4] L.M. Laita, E. Roanes-L., Verification of Knowledge Based Systems: An Algebraic Interpretation, in: Proceedings IJCAI-95 (Workshop on Verification and Validation of KBSs), McGill University, Montreal, 1995, pp. 91–95.
[5] E. Roanes-L., L.M. Laita: Un sistema de enclavamientos independiente de la topología de la estación. In: Actas del II Simposium Ingeniería del Transporte. Vol. 2. Univ. Politécnica de Madrid, 1996, pp. 409–416.
[6] E. Roanes-L., L.M. Laita, A Topology-Independent Model for Railway Interlocking Systems (Abstract), in: Proceedings of the 2nd. IMACS-ACA Conference, RISC-Linz, 1996, p. 61.
[7] J. Westwood (Ed.), Trains, Octopus, 1979.
[8] Proyecto y obra del enclavamiento electrónico de la estación de Madrid-Atocha (Proyecto Técnico), Siemens, 1988.
[9] Microcomputer Interlocking Hilversum, Siemens, 1986.
[10] Microcomputer Interlocking Rotterdam, Siemens, 1989.
[11] Puesto de enclavamiento con microcomputadoras de la estación de Chiasso de los SBB. Siemens, 1989.