ROBERT MITTERMAYR
E-mail: robert@auto.tuwien.ac.at
JOHANN BLIEBERGER
E-mail: blieb@auto.tuwien.ac.at
TU Vienna, Institute of Computer-Aided Automation
Treitlstr. 1-3, 1040 Vienna, Austria
ANDREAS SCHÖBEL
E-Mail: andreas.schoebel@tuwien.ac.at
TU Vienna, Institute of Transportation
Karlsplatz 13/230, 1040 Vienna, Austria

Traffic Planning
Original Scientific Paper
Accepted: Nov. 14, 2011
Approved: Oct. 3, 2012

KRONECKER ALGEBRA-BASED DEADLOCK ANALYSIS FOR RAILWAY SYSTEMS

ABSTRACT

Deadlock analysis for railway systems differs in several aspects from deadlock analysis in computer science. While the problem of deadlock analysis for standard computer systems is well-understood, multi-threaded embedded computer systems pose new challenges. A novel approach in this area can easily be applied to deadlock analysis in the domain of railway systems. The approach is based on Kronecker algebra. A lazy implementation of the matrix operations even allows analysing exponentially sized systems in a very efficient manner. The running time of the algorithm does not depend on the problem size but on the size of the solution. While other approaches suffer from the fact that additional constraints make the problem and its solution harder, our approach delivers its results faster if constraints are added. In addition, our approach is complete and sound for railway systems, i.e., it generates neither false positives nor false negatives.

KEY WORDS

railway networks, deadlocks, deadlock avoidance, Kronecker algebra

1. INTRODUCTION

The *deadlock* problem and its solutions were studied in the earliest days of computer science. Although computer science borrowed several concepts and namings from the railway domain such as *semaphores* or *tokens*, this was not the case for the deadlock problem. Deadlocks must have been impending in railway systems from the very beginning and railwaymen certainly were aware of them. In contrast to these facts, no solutions were commonly known in the middle of the 20th century. So computer scientists were the first to study the problem intensively and to deliver adequate solutions.

The following definition of a deadlock is given in [1]:

Deadlock: An impasse that occurs when multiple processes are waiting for the availability of a resource that will not become available because it is being held by another process that is in a similar wait state.

For a deadlock to occur, four necessary conditions were found [2]:

1. Mutual exclusion: a resource that cannot be used by more than one process at a time.
2. Hold and Wait: processes already holding resources may request new resources held by other processes.
3. No preemption: No resource can be forcibly removed from a process holding it; resources can be released only by the explicit action of the process.
4. Circular wait: two or more processes form a circular chain where each process waits for a resource that the next process in the chain holds.

If one of these conditions cannot be met, a deadlock cannot occur. Clearly all four conditions apply to railway systems.

To handle deadlock problems one distinguishes between *deadlock prevention*, *deadlock avoidance*, and *deadlock detection*.

- *Deadlock prevention* tries to remove one of the four conditions above in order to make it impossible for a deadlock to occur.
- *Deadlock avoidance* utilizes certain information about processes known in advance to decide whether or not to allocate resources to processes. Examples for deadlock avoidance algorithms include the Banker's algorithm [3], Wait/Die, Wound/Wait algorithms, and [4] to name a few.

– When a deadlock is *detected* at runtime, a process is terminated and restarted later on.

Deadlocks in railway systems attracted attention when railway simulation systems became available. Since simulations were supposed to run without human interaction, deadlocks became an obstacle. Obviously, autonomous dispatching systems also suffer from deadlock problems if no adequate solutions can be found.

The outline of our paper is as follows. In Section 2 we contrast deadlocks in computer systems with deadlocks in railway systems. Section 3 introduces Kronecker algebra. Section 4 gives our standard system model. In Section 5 we present a simple example. Section 6 shows how our resulting graphs can be employed for finding optimal solutions. In Section 7 we describe our lazy implementation of Kronecker operations. In Section 8 we extend our standard model in order to solve several important practical problems. In Section 9 we present experimental data. In Section 10 we relate our approach to other work, before we conclude our paper in Section 11.

2. DEADLOCKS IN COMPUTERS VS. DEADLOCKS IN RAILWAY SYSTEMS

In railway systems trains, routes, and track sections correspond to processes, program code, and shared resources in computer systems, respectively.

The table below shows major differences between the deadlock problem in computer and railway systems.

From the table below it becomes obvious that deadlock analysis methods and algorithms well-suited for standard computer systems cannot be simply transferred to railway systems. However, safety-related embedded systems show more resemblance to railway system. This is the reason why we try to apply the approach we have developed for embedded systems [4] to railway systems. We will show in this paper that this approach is in fact well-suited for railway systems.

In the following we will argue that only deadlock avoidance makes sense in the domain of railway systems; deadlock prevention and deadlock detection cannot be applied in this domain.

Deadlock prevention is done by ensuring that one of the four necessary conditions stated above is not fulfilled. It is easy to see that this cannot be done for railway systems:

1. Mutual exclusion: Track sections have to be seized by trains exclusively.
2. Hold and Wait: It is typical that a train occupies two or more track sections at the same time.
3. No preemption: A track section cannot be removed from a train "holding" it.
4. Circular wait: It is impossible to impose a strict order on the track sections without posing the problem that some trains would seize all track sections of their route from start to destination before they start running.

Deadlock detection is of no interest for railway systems because "terminating a train" is no option.

The only remaining option is *deadlock avoidance* which will be discussed in the rest of this paper.

Before we give an introduction to Kronecker algebra, we state a few additional differences between computer and railway systems:

– Computer systems usually consist of a number of processes (threads) and a number of shared resources. In most cases the number of resources is much smaller than the number of processes. In railway systems the number of track sections is usually much greater than the number of trains.
– Besides the track sections being shared resources in railway systems there are additional constraints that have to be met. For example, connections and overtaking have to be taken care of, tardy trains may cause penalty, ...

While previous approaches perform poorly if additional constraints are added to the problem domain,

Computer systems	Railway systems
In standard computer applications all resources are held until the process terminates (not true for embedded systems where processes may not terminate at all).	A train seizes a track section shortly before it enters it, and holds it until it leaves it (which may be long before it reaches its destination).
Program code consists of straight line code, if, and loop statements.	Routes compare to straight line code; there may be alternative routes which compare to if statements; loops may be useful for model railroad applications.
If a multithreaded computer program contains a deadlock, the program is erroneous.	If in a railway system it is possible to bypass a deadlock situation, then the system is correct. Deadlocks are *always possible*, but can usually be bypassed.
A computer program is "correct" if we can prove it deadlock-free.	A set of trains is schedulable if all possible deadlocks can be bypassed.
If a deadlock is detected, it is viable to terminate a program and restart it later on (not true for safety-related embedded systems).	Trains cannot simply be taken out of the overall system.

our approach delivers its results faster if constraints are added. This makes it a very promising approach. Details will be given in the following sections.

3. KRONECKER ALGEBRA – AN OVERVIEW

Kronecker product and Kronecker sum form Kronecker algebra. In the following we define both operations, state properties, and give examples.

We define the set of matrices

$$\mathcal{M} = \{M = (m_{i,j}) \mid m_{i,j} \in \mathcal{L}\}$$

where $\mathcal{L}$ denotes a set of labels such that $\langle \mathcal{L}, +, 0 \rangle$ is a commutative monoid and $\langle \mathcal{L}, \cdot, 1 \rangle$ is a monoid. In the remaining parts of this paper only matrices $M \in \mathcal{M}$ will be used, except where stated explicitly. Let $o(M)$ refer to the order[1] of matrix $M \in \mathcal{M}$. In addition we will use n-by-n zero matrices $Z_n = (z_{i,j})$, where $\forall i,j: z_{i,j} = 0$.

Definition 1 (Kronecker product). Given an m-by-n matrix A and a p-by-q matrix B, their Kronecker product - denoted by $A \otimes B$ - is an mp-by-nq block matrix defined by

$$A \otimes B = \begin{pmatrix} a_{1,1}B & \cdots & a_{1,n}B \\ \vdots & \ddots & \vdots \\ a_{m,1}B & \cdots & a_{m,n}B \end{pmatrix}.$$

Example 1:

Let

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \text{ and } B = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}.$$

Kronecker product $C = A \otimes B$ is given by

$$\begin{pmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,1}b_{1,3} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} & a_{1,2}b_{1,3} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,1}b_{2,3} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} & a_{1,2}b_{2,3} \\ a_{1,1}b_{3,1} & a_{1,1}b_{3,2} & a_{1,1}b_{3,3} & a_{1,2}b_{3,1} & a_{1,2}b_{3,2} & a_{1,2}b_{3,3} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,1}b_{1,3} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} & a_{2,2}b_{1,3} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,1}b_{2,3} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} & a_{2,2}b_{2,3} \\ a_{2,1}b_{3,1} & a_{2,1}b_{3,2} & a_{2,1}b_{3,3} & a_{2,2}b_{3,1} & a_{2,2}b_{3,2} & a_{2,2}b_{3,3} \end{pmatrix}.$$

As stated in [5] the Kronecker product is also being referred to as *Zehfuss product or direct product of matrices or matrix direct product*.[2]

Next we list some basic properties of the Kronecker product. Proofs and additional properties can be found in [6], [7], [8], [9]. Let A, B, C, and D be matrices. The Kronecker product is non-commutative because in general $A \otimes B \neq B \otimes A$. It is permutation equivalent because there exist permutation matrices P and Q such that $A \otimes B = P(B \otimes A)Q$. If A and B are square matrices, then $A \otimes B$ and $B \otimes A$ are even permutation similar, i.e., $P = Q^T$. The product is associative as

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C.$$

In addition, the Kronecker product distributes over +, i.e.

$$A \otimes (B + C) = A \otimes B + A \otimes C \text{ and}$$

$$(A + B) \otimes C = A \otimes C + B \otimes C.$$

The Kronecker product can be used to model synchronisation. How this is done will be discussed later.

Definition 2 (Kronecker sum). Given matrix A of order m and matrix B of order n, their Kronecker sum denoted by $A \oplus B$ is a matrix of order mn defined by

$$A \oplus B = A \otimes I_n + I_m \otimes B,$$

where I_m and I_n denote identity matrices[3] of order m and n, respectively.

This operation must not be confused with the direct sum of matrices, group direct product or direct product of modules for which the symbol $\oplus$ is used, too.

Example 2:

We use matrices A and B from Example 1. The Kronecker sum $A \oplus B$ is given by

$$A \otimes I_3 + I_2 \otimes B =$$

$$= \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} =$$

$$= \begin{pmatrix} a_{1,1} & 0 & 0 & a_{1,2} & 0 & 0 \\ 0 & a_{1,1} & 0 & 0 & a_{1,2} & 0 \\ 0 & 0 & a_{1,1} & 0 & 0 & a_{1,2} \\ a_{2,1} & 0 & 0 & a_{2,2} & 0 & 0 \\ 0 & a_{2,1} & 0 & 0 & a_{2,2} & 0 \\ 0 & 0 & a_{2,1} & 0 & 0 & a_{2,2} \end{pmatrix} +$$

$$+ \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & 0 & 0 & 0 \\ b_{2,1} & b_{2,2} & b_{2,3} & 0 & 0 & 0 \\ b_{3,1} & b_{3,2} & b_{3,3} & 0 & 0 & 0 \\ 0 & 0 & 0 & b_{1,1} & b_{1,2} & b_{1,3} \\ 0 & 0 & 0 & b_{2,1} & b_{2,2} & b_{2,3} \\ 0 & 0 & 0 & b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} =$$

$$= \begin{pmatrix} a_{1,1} + b_{1,1} & b_{1,2} & b_{1,3} & a_{1,2} & 0 & 0 \\ b_{2,1} & a_{1,1} + b_{2,2} & b_{2,3} & 0 & a_{1,2} & 0 \\ b_{3,1} & b_{3,2} & a_{1,1} + b_{3,3} & 0 & 0 & a_{1,2} \\ a_{1,2} & 0 & 0 & a_{2,2} + b_{1,1} & b_{1,2} & b_{1,3} \\ 0 & a_{1,2} & 0 & b_{2,1} & a_{2,2} + b_{2,2} & b_{2,3} \\ 0 & 0 & a_{1,2} & b_{3,1} & b_{3,2} & a_{2,2} + b_{3,3} \end{pmatrix}.$$

In the following we list basic properties of the Kronecker sum of matrices A, B, and C. Additional properties can be found in [10]. The Kronecker sum is non-commutative because for element-wise comparison in general $A \oplus B \neq B \oplus A$. Anyway, it essentially commutes because from a graph point of view, the graphs represented by matrices $A \oplus B$ and $B \oplus A$ are structurally isomorphic. In addition, we state a property of the Kronecker sum which we call *Mixed Sum Rule*.

Let the matrices A and C have order m and B and D have order n. Then we have

$$(A \oplus B) + (C \oplus D) = (A + C) \oplus (B + D).$$

A proof of the Mixed Sum Rule can be found in [4].

In addition, the Kronecker sum is associative (cf. [4]), i.e.,

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C.$$

The associativity properties of the operations $\otimes$ and $\oplus$ imply that the k-fold operations

$$\otimes_{i=1}^{k} A_i \text{ and } \oplus_{i=1}^{k} A_i$$

are well-defined.

The Kronecker sum calculates all possible interleavings (see e.g. [11] for a proof). The following example illustrates interleaving and how the Kronecker sum handles it.

Example 3: Let matrices C and D be defined as follows:

$$C = \begin{pmatrix} 0 & a & 0 \\ 0 & 0 & b \\ 0 & 0 & 0 \end{pmatrix}, \; D = \begin{pmatrix} 0 & c & 0 \\ 0 & 0 & d \\ 0 & 0 & 0 \end{pmatrix}$$

As already mentioned above, there is a correspondence between matrices and *graphs*. In general, a directed labelled graph $G\langle V, E, n_e \rangle$ consists of a set of labelled nodes V, a set of labelled directed edges $E \subseteq V \times V$, and an entry node n_e. In this paper we label the graph nodes simply by positive integers. The correspondence between graphs and matrices – frequently called *adjacency matrices* – is as follows. If there exists an edge labelled a from node i to node j, then the corresponding adjacency matrix M has $m_{i,j} = a$. If there is no edge between node i and node j, then $m_{i,j} = 0$. The graph corresponding to matrix C from above is depicted in *Figure 1*, whereas the graph of matrix D is shown in *Figure 2*. Now interpret C and D as being trains and a, b, c and d as being actions of the trains with the following meaning: a denotes train C enters track section T_a, b means train C has left track section T_a, c denotes train D enters track section T_b, and d means train D has left track section T_b. All possible temporal interleavings of these actions are shown in *Table 1*. In *Figure 3* the graph represented by the adjacency matrix $C \oplus D$ is depicted. It is easy to see that all possible temporal interleavings are generated correctly.

Table 1

Interleavings
$a \cdot b \cdot c \cdot d$
$a \cdot c \cdot b \cdot d$
$a \cdot c \cdot d \cdot b$
$c \cdot a \cdot b \cdot d$
$c \cdot a \cdot d \cdot b$
$c \cdot d \cdot a \cdot b$

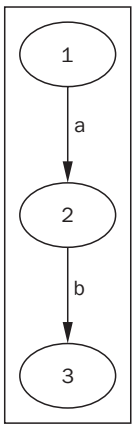

Figure 1

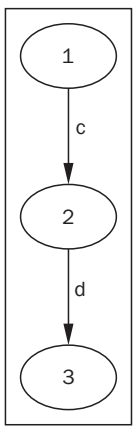

Figure 2

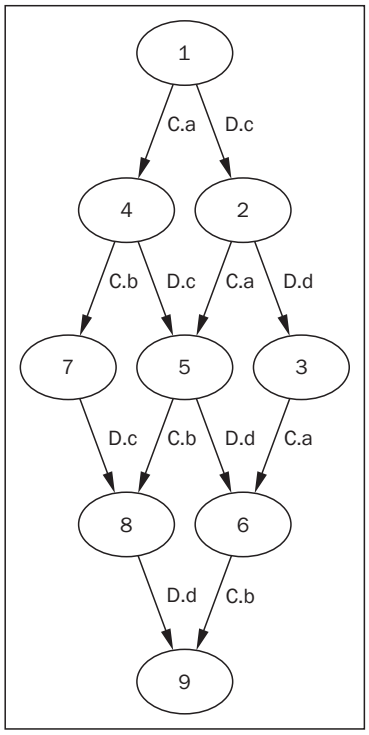

Figure 3

In general, by calculating the Kronecker sum of the adjacency matrices of two graphs the adjacency matrix of the Cartesian product graph [12] is computed (cf. [13]).

Now assume that T_a and T_b denote the same track section. It is clear that in this case the temporal interleavings of *Table 1* are no more valid. The trains have to *synchronise* in order to perform their actions correctly. This can be modelled by Kronecker product and an additional matrix of the form

$$S = \begin{pmatrix} 0 & p \\ v & 0 \end{pmatrix}$$

where p denotes the action "Enter the track section" and v means "Train has left the track section". The correct system behaviour can be described by matrix $R = (C \oplus D) \otimes S$. The corresponding graph is shown in *Figure 4*. It is interesting that the graph became decomposed into seven parts (sub-graphs). Clearly, only the part reachable from the entry node is responsible for the system behaviour, the other six parts can safely be ignored. Thus we concentrate on the sub-graph with entry node 1. Studying this sub-graph we see that now the trains enter the track section one after the other. Note that the two paths in the subgraph correctly mirror the two cases where C enters the track section before D and vice versa. A proof that Kronecker product models synchronisation correctly can be found in [4]. It is also worth noting, that parts unreachable from the entry node always occur if synchronisation via Kronecker product takes place. This observation gives rise to the lazy implementation described in Section 7.

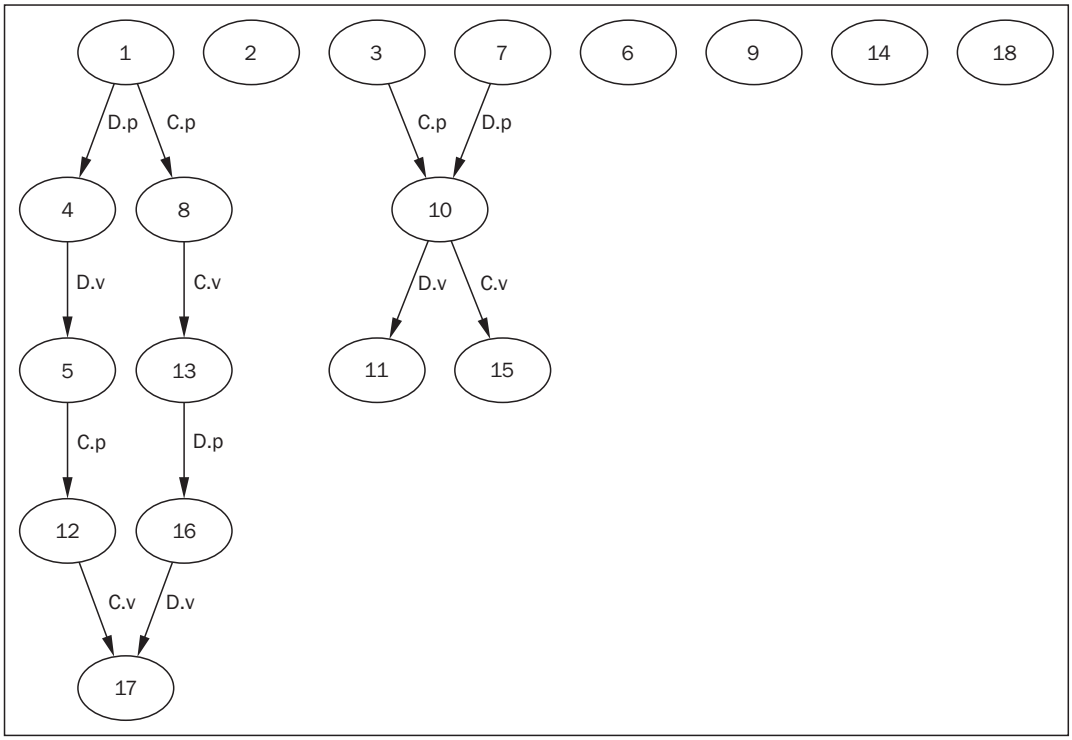

Figure 4

4. SYSTEM MODEL

We model a general railway system S by a set of track sections $T = \{T_i \mid 1 \le i \le r\}$. Each track section T_i is modelled by matrix

$$T_i = \begin{pmatrix} 0 & p_i \\ v_i & 0 \end{pmatrix}.$$

In addition, a railway system consists of a set of trains $L = \{L_j \mid 1 \le j \le t\}$. The route R_j of train L_j is a sequence of track sections $T_{l_1}, \ldots, T_{l_s}$ for $1 \le l_n \le r$ and $1 \le n \le s$. Each route is modelled by a $2s \times 2s$-matrix. In particular, $R_j = (r_{i,k})$. The set of routes is denoted by $R = \{R_j \mid 1 \le j \le t\}$.

The behaviour of railway system $S\langle T, L, R\rangle$ is modelled by

$$S = (\oplus_{j=1}^{t} R_j) \otimes (\oplus_{i=1}^{r} T_i),$$

where during the evaluation of the Kronecker product we let $p_i = p_i \cdot p_i$ and $v_i = v_i \cdot v_i$.

The different paths in the graph corresponding to matrix S mirror *all possible behaviours* of the railway system in terms of temporal interleavings of the actions of trains, namely entering and leaving track sections.

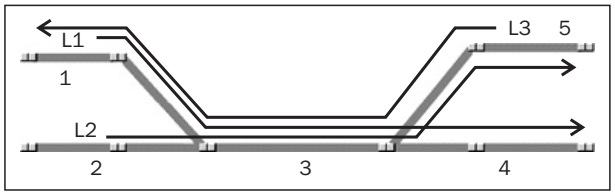

Figure 5

From the discussion above and from [4] it is clear that deadlocks appear as purely structural properties of the underlying graph. Deadlocks manifest themselves as non-final nodes[4] with no successors.

5. A SIMPLE EXAMPLE

In this section we give a small example on how deadlocks can be avoided by our Kronecker algebra-based approach.

In *Figure 5* a typical scenario is shown that may lead to a deadlock. The routes of the three involved trains are given in *Table 2*. If train L_3 enters track section T_3 before the other trains move, a deadlock is unavoidable.

Table 2

Trains	Routes
L_1	p_3, v_1, p_4, v_3, v_4
L_2	p_3, v_2, p_5, v_3, v_5
L_3	p_3, v_5, p_1, v_3, v_1

The matrices for the three routes are set up as follows:

$$R_1 = \begin{pmatrix} 0 & p_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & v_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & p_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & v_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & v_4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad R_2 = \begin{pmatrix} 0 & p_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & v_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & p_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & v_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & v_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\text{and } R_3 = \begin{pmatrix} 0 & p_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & v_5 & 0 & 0 & 0 \\ 0 & 0 & 0 & p_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & v_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & v_1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The matrices for the five track sections have the form

$$T_i = \begin{pmatrix} 0 & p_i \\ v_i & 0 \end{pmatrix}$$

for $1 \leq i \leq 5$.

The order of the resulting matrix can be computed by multiplying the order of the involved matrices, i.e., the order of matrices R_j and the order of matrices T_i. In our example the resulting matrix will have order $6912 = 6 \cdot 6 \cdot 6 \cdot 2^5$. Due to synchronisation only a small part of the resulting graph is reachable from the entry node. Only this part – consisting of 42 nodes – is displayed in *Figure 6*. To increase readability our implementation distinguishes between the following node types:

- Diamond nodes denote deadlocks or nodes from which only deadlocks can be reached.
- Solid nodes denote *safe states*. A state is *safe* if all trains can perform their actions without having to take into account the moves of other trains in the system, provided that the track section which they are to enter is not occupied by another train.[5]
- From dotted nodes both diamond and solid nodes can be reached.

If only deadlock avoidance is of interest, graphs like that in *Figure 6* can be scaled down. In fact, diamond nodes are of no interest because we want to avoid deadlocks. From the set of solid nodes only those are of interest, which have dotted predecessors only. In addition, we do not have to show the node labels, i.e., the matrix line numbers. If we eliminate the non-interesting nodes from *Figure 6*, we obtain the graph in *Figure 7*. We call such graphs *NDLS-graphs* because they contain **n**o **d**eadlock nodes and only a **l**imited number of **s**afe nodes.

In order to avoid deadlocks the minimal number of actions can be obtained by selecting a path from the entry node to a solid node which is situated nearest to the entry node. This, however, may not be the best alternative. In the next section we elaborate on the problem of finding the "cheapest" path.

In order to evaluate the quality of algorithms or tools it is convenient to introduce *false positives* and *false negatives*. Let us assume we have an algorithm at hand that examines a system to find out whether the system deadlocks or not. Then there are four possible outcomes.

	Algorithm says system deadlocks	Algorithm says system does not deadlock
System deadlocks	okay	False negative
System does not deadlock	False positive	okay

It is worth noting that our approach does neither deliver false positives nor false negatives[6]. Thus it is

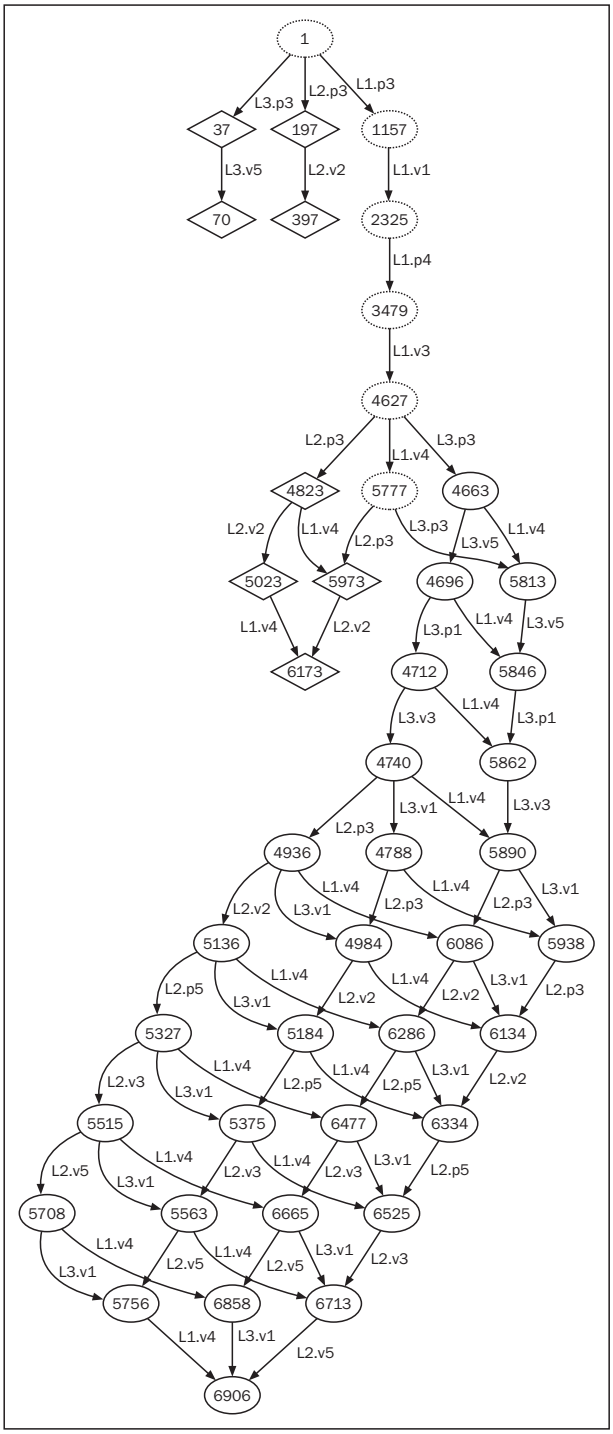

Figure 6

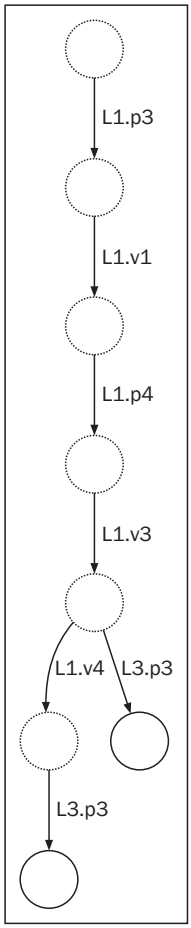

Figure 7

complete and *sound*. There is only one exception: In case of alternative routes (cf. Section 8) deadlocks in all possible paths will be reported although only one path can be chosen by the train. This situation may somehow be classified as reporting false positives.

For computer systems our approach can be modified such that it stops as soon as the first deadlock is found[7]. For railway systems the algorithm may be modified to stop as soon as a safe state is found.

6. AVOIDING DEADLOCKS UNDER ADDITIONAL CONSTRAINTS

Our approach produces a graph which gives all temporal interleavings of train moves. A short reflection shows that not all of these interleavings may result in an "optimal" behaviour. However, a more sophisticated analysis may choose to assign weights to the edges of the graph. These weights can be chosen such that the term "optimal" from above makes sense. Weights may include temporal constraints originating from physical properties of the trains such as accelerating and braking characteristics depending e.g. on the weight of the train, or penalties due to tardiness.

In any case, well-known algorithms such as those of [14] or [15] can be applied to graphs with non-negative weights to find the optimal (shortest) path. Dijkstra's algorithm performs in time quadratic in the number of nodes of the graph. The Fredman & Tarjan version uses time proportional to the sum of the number of edges and the number of nodes times the logarithm of the number of nodes.

If only deadlock avoidance is of concern, the optimisations given in the previous section apply.

7. LAZY IMPLEMENTATION OF KRONECKER ALGEBRA

Until now we primarily focused on a pure mathematical model for deadlock analysis. An alert reader will have noticed that the order of the matrices increases exponentially in the number of trains. On the other hand, we have seen that the Kronecker product results in parts of the resulting matrix that cannot be reached from the entry node of the corresponding graph. This comes solely from the fact that synchronisation excludes some interleavings. In addition, all involved matrices are sparse (cf. [4]), such that space saving data structures suggest themselves.

Choosing a lazy implementation [16] for the matrix operations, however, ensures that, when extracting the reachable parts of the underlying graph, the overall effort is reduced to exactly these parts. By starting from the entry node and calculating all reachable successor nodes, our lazy implementation does exactly this. Thus, for example, if the resulting graph's size is linear in terms of the involved trains, only linear effort will be necessary to generate the resulting graph.

Our implementation distinguishes between two kinds of matrices: Sparse matrices are used for representing trains and track sections. Lazy matrices are employed for representing all the other matrices, i.e. those resulting from the operations of the Kronecker algebra. Besides the employed operation, a lazy matrix simply keeps track of its operands. Whenever an entry of a lazy matrix is retrieved, depending on the operation recorded in the lazy matrix, entries of the operands are retrieved and the recorded operation is performed on these entries to calculate the result. In the course of this computation, even the successors of nodes are evaluated lazily. Retrieving entries of operands is done recursively if the operands are again lazy matrices, or it is done by retrieving the entries from the sparse matrices, where the actual data resides.

In addition, our lazy implementation allows for simple parallelising. For example, retrieving the entries of left and right operands can be done concurrently. Exploiting this, we expect further performance improvements for our implementation if run on multicore architectures. All examples in this paper have

been computed by our lazy implementation. The performance of our implementation is discussed in Section 9.

8. EXTENSIONS OF THE STANDARD MODEL

In this section we discuss how our standard model defined in Section 4 can be extended in order to solve certain important problems.

The first problem we consider occurs if the length of a train is greater than the length of a track section. Assume a route of a train containing three consecutive track sections t_1, t_2, and t_3, where t_2 is shorter than the length of the train. In addition, assume that the train occupies track section t_1. If t_2 were long enough, the train's moves would be $p_2, v_1, p_3, v_2, \ldots$ In case of the short track section the train's moves are $p_2, p_3, v_1, v_2, \ldots$

It is clear that this approach can easily be extended if more than three track sections are used by a train simultaneously.

Furthermore, note that double slip switches can be seen as two normal switches with a zero length track section in between. Thus, routes containing double slip switches represent typical examples of the method above.

Sometimes it is useful to model a railway system from a more *abstract view*. For example one abstracts away from the track details of a station and instead would like to use only the fact that the station has a capacity of holding c trains at the same time, where each train is able to enter and leave the station independently from the other trains in the station. Such a station can be modelled by a $(c+1)\times(c+1)$ matrix of the following form

$$\begin{pmatrix} 0 & p & 0 & \cdots & 0 \\ v & 0 & p & \cdots & 0 \\ 0 & v & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & v & 0 & p \\ 0 & 0 & \cdots & v & 0 \end{pmatrix}.$$

Thus, our approach is able to handle abstract views as well. One can envisage even more abstract levels such as abstracting whole railway lines to their capacity. This allows to model nets of railway lines.

Sometimes it is useful when a train is allowed to take alternative routes. This gives more freedom in bypassing a potential deadlock situation. Since our approach was developed for handling embedded computer systems, it is able to handle alternatives which are present as if-statements in computer programs. Generating matrices that model alternative routes is very easy. A line in the matrix representing a fork state of a route has two successors, i.e., two non-zero entries. Later on the different alternatives can be joined and a single (straightforward) route continues. Clearly, this works even for more than two alternative routes.

As already noted, our approach was developed for embedded computer systems. Thus, it also supports loop statements. Matrices modelling loops can be employed to model trains running in loops. This might be of interest for model railroaders.

By adding further matrices we can model synchronisation of trains which can be used to ensure connections and overtaking of trains. For example, assume that we want to ensure that train A shall not leave track section t_A before train B leaves track section t_B. We introduce two "artificial track sections" d and e for synchronisation with the usual operations p_d, v_d, p_e and v_e. In addition, we insert immediately in front of the moves v_A and v_B the actions v_d, p_e and p_d, v_e, respectively. Thus, the two trains are synchronised and we are able to model connections correctly.

We conclude this section with a more elaborate example containing some of the extensions described above. In particular, it contains a double slip switch (track section 7) and it shows how overtaking can be modelled.

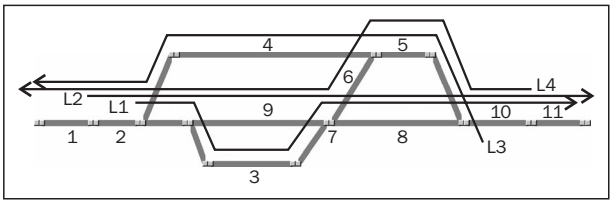

Figure 8

The system is depicted in *Figure 8*. The routes are defined as follows:

$R_1 = p_3, v_2, p_7, p_8, v_3, v_7, p_{10}, v_8, p_{11}, v_{10}, v_{11}$,
$R_2 = p_2, v_1, p_9, v_2, p_7, p_8, v_9, v_7, p_{10}, v_8, p_{11}, v_{10}, v_{11}$,
$R_3 = p_5, v_{10}, p_4, v_5, p_2, v_4, p_1, v_2, v_1$, and
$R_4 = p_{10}, v_{11}, p_5, v_{10}, p_6, v_5, p_7, p_9, v_6, v_7, p_2, v_9, p_1, v_2, v_1$.

Note that track section 7 has zero length; it originates from the double-slip switch located between track sections 3, 9, 6, and 8.

Kronecker algebra calculations produce a graph with 55,050,240 potential nodes. After the reductions presented in Section 5 only 992 nodes are left.

Now we introduce additional constraints. We assume that train L_2 overtakes L_1 and L_4 overtakes L_3 within the station. We need one additional "artificial track section" for each train pair. The corresponding operations are denoted by p_{12}, v_{12}, p_{13}, and v_{13}. Now the routes read as follows:

$R_1 = p_3, v_2, \mathbf{v_{12}}, p_7, p_8, v_3, v_7, p_{10}, v_8, p_{11}, v_{10}, v_{11}$
$R_2 = p_2, v_1, p_9, v_2, p_7, p_8, v_9, v_7, p_{10}, v_8, \mathbf{p_{12}}, p_{11}, v_{10}, v_{11}$
$R_3 = p_5, v_{10}, p_4, v_5, \mathbf{v_{13}}, p_2, v_4, p_1, v_2, v_1$, and
$R_4 = p_{10}, v_{11}, p_5, v_{10}, p_6, v_5, p_7, p_9, v_6, v_7, p_2, v_9, \mathbf{p_{13}}, p_1, v_2, v_1$.

Kronecker algebra produces 298,721,280 potential nodes. After the usual reductions, however, only

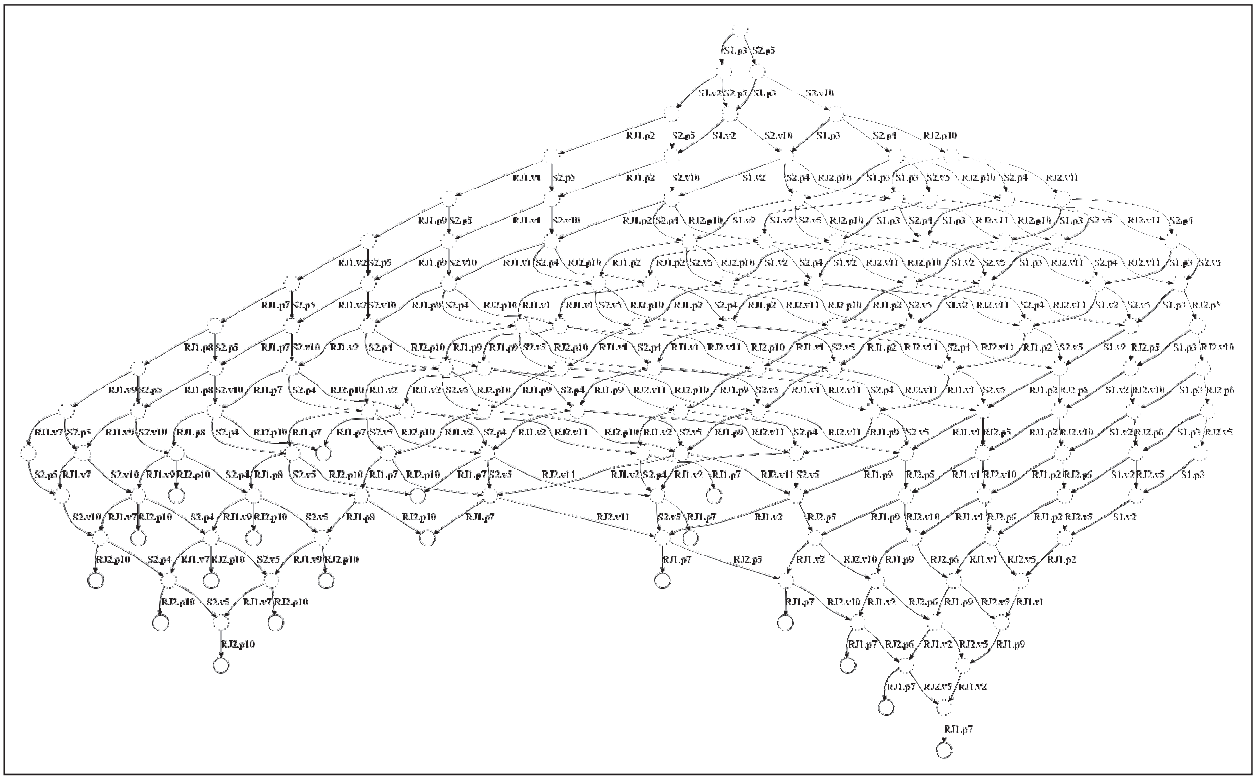

Figure 9

144 nodes are left (cf. *Figure 9*). This means that the number of potential nodes has grown by a factor of five, but the size of the solution has dropped by factor of approximately 6.8. The resulting NDLS-graph is depicted in *Figure 9*.

This example shows that if constraints are added to a railway system, our approach delivers its solution faster. This is rather atypical for recent approaches.

9. EXPERIMENTAL DATA

In this section we give experimental data for ten examples. Let $o(S)$ and $o(C)$ refer to the order of the adjacency matrix S, which is *not* computed by our lazy implementation, and the order of the adjacency matrix C of the resulting graph, respectively. $o(NDLS)$ denotes the order of the NDLS graph. In addition, k and r refer to the number of trains and the number of track sections, respectively.

Next we refer to the data depicted in *Table 3*. The runtime of our implementation shows a strong correlation to order $o(C)$ of the adjacency matrix C of the resulting graph with a Pearson product-moment correlation coefficient of 0.999027713. The correlation coefficient between $o(NDLS)$ and the runtime is 0.9813607056. In contrast, the values of the purely theoretical order $o(S)$ of the resulting adjacency matrix S correlate to the runtime only with a correlation coefficient of 0.2370050995.

These observations show that the runtime complexity does not depend on the order $o(S)$ which grows exponentially in the number of trains. We conclude this section by stating that the collected data give strong indication that the runtime complexity of our approach is linear in the number of nodes present in the resulting graph.

Table 3

k	r	$o(S)$	$o(C)$	$o(NDLS)$	Runtime [s]
2	4	256	12	1	0.03
3	5	4800	30	8	0.097542
4	6	124416	98	20	0.48655
3	6	75264	221	32	1.057529
4	7	614400	338	48	2.537082
4	8	1536000	277	94	2.566587
4	8	737280	380	116	3.724364
4	13	298721280	2583	206	96.024073
4	11	55050240	3908	992	146.81
5	6	14929920	7666	1812	309.371395

10. RELATED WORK

Concerning Kronecker algebra probably the closest work to ours was done by Buchholz and Kemper [17]. It differs from our work in that we use our approach to avoid deadlocks. Buchholz and Kemper

worked on generating reachability sets in composed automata and for describing networks of synchronized automata. Both approaches employ Kronecker algebra.

In [18] we used similar definitions of Kronecker algebra. The approach is used for analysing the timing behaviour of concurrent programs.

Although not closely related we recognize the work done in the field of stochastic automata networks (SAN) which is based on the work of [19] and in the field of generalized stochastic Petri nets (GSPN) (e.g. [20]) as related work. Compared to ours these fields are completely different. Nevertheless, basic operators are shared and some properties influenced this paper.

In [21] Banker's algorithm [3] is modified such that it can be employed for deadlock analysis in railway systems. Since Banker's algorithm has been designed for standard computer systems it is not well-suited for railway systems. For example, it may prohibit allocating a resource (track section) although a potential deadlock can be bypassed. In contrast to our approach both track sections and switches have to be modelled as resources in [21].

An operations research approach is proposed in [22] to do deadlock analysis in railroad systems.

In [23] Movement Consequence Analysis (MCA) and Dynamic Route Reservation (DRR) are introduced for deadlock analysis. Both are rule-based methods for which correctness cannot be proved. It delivers false positives.

An algorithm which can handle only simple railway networks is given in [24]. In addition, it produces false positives. The same restrictions apply to the approach presented in [25].

Deadlock-free algorithms are presented in [26]. All of them may generate false positives.

A coloured Petri net model is used in [27] to describe a railway network system and to derive the traffic controller. Safeness and deadlock freedom are guaranteed. A deadlock prevention strategy is defined and expressed by a set of linear inequality constraints. It is shown how collision and deadlock prevention constraints can be expressed as coloured generalised mutual exclusion constraints and that the controller can be implemented by a set of monitor places. It is however not clear how this approach can be extended to find optimal solutions such as our approach suggests (cf. Section 6).

In [28] a coloured Petri net model of a simple railway station operation is constructed and Banker's algorithm is employed for deadlock avoidance. It turned out that Banker's algorithm is not able to handle a large number of processes (trains) and resources (track sections).

11. CONCLUSION

We have presented a Kronecker algebra-based approach for deadlock analysis in railway systems. It can solve all important practical problems. By employing a lazy implementation for the Kronecker matrix operations the solution is computed very efficiently.

By assigning weights to the edges of the resulting graph various optimisations can be performed. This includes optimisations due to physical properties of the trains such as accelerating and braking characteristics depending e.g. on the weight of the train, or penalties due to tardiness.

The running time of the algorithm does not depend on the problem size but on the size of the solution. While other approaches suffer from the fact that additional constraints make the problem and its solution harder, our approach delivers its results faster if constraints are added.

ROBERT MITTERMAYR
E-mail: robert@auto.tuwien.ac.at
JOHANN BLIEBERGER
E-mail: blieb@auto.tuwien.ac.at
TU Wien, Institut für rechnergestützte Automation
Treitlstr. 1-3, 1040 Wien, Österreich
ANDREAS SCHÖBEL
E-mail: andreas.schoebel@tuwien.ac.at,
TU Wien, Institut für Verkehrswissenschaften
Karlsplatz 13/230-2, 1040 Wien, Österreich

ZUSAMMENFASSUNG

KRONECKER ALGEBRA BASIERENDE DEADLOCK-ANALYSE VON EISENBAHN-DISPOSITIONSSYSTEMEN

Deadlock-Analyse für Eisenbahn-Dispositionssysteme unterscheidet sich von Deadlock-Analyse in der Informatik in mehreren Punkten. Während Deadlock-Analyse für Standard-Computersysteme im Allgemeinen gut verstanden ist, stellen Eingebettete Systeme eine neue Herausforderung dar. Ein neuer Ansatz aus diesem Bereich kann auf Eisenbahn-Dispositionssysteme angewandt werden. Der Ansatz basiert auf Kronecker Algebra. Eine „Lazy Implementierung" der Matrixoperationen erlaubt sogar die Analyse von exponentiell wachsenden Systemen in sehr effizienter Art und Weise.

Die Laufzeit des Algorithmus hängt nicht von der Problemgröße sondern von der Größe des Ergebnisses ab. Während andere Ansätze durch zusätzliche Einschränkungen langsamer werden, liefert unsere Methode das Ergebnis schneller, sobald weitere Einschränkungen hinzukommen. Weiters ist unser Ansatz vollständig (complete) und fehlerfrei (sound), d.h. es treten weder „false positives" noch „false negatives" auf.

SCHLÜSSELWÖRTER

Eisenbahnnetze, Deadlocks, Deadlock-Vermeidung, Kronecker Algebra

REFERENCES

1. A k-by-k matrix is known as square matrix of order *k*.
2. Knuth notes in [13] that Kronecker never published anything about it. Zehfuss was actually the first publishing it in the 19th century [29]. He proved that $\det(A \otimes B) = \det^n(A) \cdot \det^m(B)$, if *A* and *B* are matrices of order *m* and *n*, respectively, with entries from the domain of real numbers.
3. The identity matrix I_n is an n-by-n matrix with ones on the main diagonal and zeros elsewhere.
4. A *final node* corresponds to the destination of a route. A final node of a railway system corresponds to the state, where all trains have reached their destinations.
5. If a track section is occupied by another train, the movement of the train wanting to enter may be delayed, but no deadlock can occur.
6. This is true for railway systems. It is not true for computer systems because computer programs may e.g. contain *dead code*, i.e., code that is never executed. Our approach will report a deadlock even if it is contained in dead code. Thus it will deliver false positives for computer systems but no false negatives.
7. This makes sense because a computer system is considered erroneous if at least one deadlock is present.

LITERATURE

[1] **Stallings, W.**: *Operating Systems, 4th Ed.*, Upper Saddle River, New Jersey: Prentice-Hall, 2001

[2] **Coffman, E. G. J.**, **Elphick, M. J.** and **Shoshani, A.**: "*System deadlocks*," ACM Computing Surveys, Vol. 3, No. 2, pp. 67-78, 1971

[3] **Dijkstra, E. W.**: "*Een algorithme ter voorkoming van de dodelijke omarming* (EWD-108)".

[4] **Mittermayr, R.** and **Blieberger, J.**: "*Shared Memory Concurrent System Verification using Kronecker Algebra*", 2011. [Online]. Available: http://arxiv.org/abs/1109.5522.

[5] **Miller, J.**: "*Earliest Known Uses of Some of the Words of Mathematics*", 2011. [Online]. Available: http://jeff560.tripod.com/k.html. [Accessed 26 Sept. 2011]

[6] **Bellman, R.**: *Introduction to Matrix Analysis, Classics in Applied Mathematics*, 2nd Ed., Society for Industrial and Applied Mathematics, 1997

[7] **Graham, A.**: *Kronecker Products and Matrix Calculus with Applications*, New York: Ellis Horwood Ltd., 1981

[8] **Davio, M.** "*Kronecker Products and Shuffle Algebra*", IEEE Trans. Computers, Vol. 30, No. 2, pp. 116-125, 1981

[9] **Hurwitz, A.**: "*Zur Invariantentheorie*", Math. Annalen, Vol. 45, pp. 381-404, 1894

[10] **Plateau, B.** and **Atif, K.**: "*Stochastic Automata Network For Modeling Parallel Systems*", IEEE Trans. Software Eng., Vol. 17, No. 10, pp. 1093-1108, 1991

[11] **Küster, G.**: "*On the Hurwitz Product of Formal Power Series and Automata*", Theor. Comput. Science, Vol. 83, No. 2, pp. 261-273, 1991

[12] **Imrich, W.**, **Klavzar, S.** and **Rall, D. F.**: *Topics in Graph Theory: Graphs and Their Cartesian Product*, A K Peters Ltd, 2008

[13] **Knuth, D. E.**: *Combinatorial Algorithms*, The Art of Computer Programming, Vol. 4A, Addison-Wesley, 2011

[14] **Dijkstra, E. W.**: "*A note on two problems in connexion with graphs*", Numerische Mathematik, Vol. 1, pp. 269-271, 1959

[15] **Fredman, M. L.** and **Tarjan, R. E.**: "*Fibonacci heaps and their uses in improved network optimization algorithms*", in 25th Annual Symposium on Foundations of Computer Science, 1984

[16] **Henderson, P.** and **Morris, J. J. H.**: "*A Lazy Evaluator*", in Proceedings of the 3rd ACM SIGACT-SIGPLAN symposium on Principles on programming languages, 1976

[17] **Buchholz, P.** and **Kemper, P.**: "*Efficient Computation and Representation of Large Reachability Sets for Composed Automata*", Discrete Event Dynamic Systems, Vol. 12, No. 3, pp. 265-286, 2002

[18] **Mittermayr, R.** and **Blieberger, J.**: *Timing Analysis of Concurrent Programs*, In Proc. of 12th Int. Workshop on Worst-Case Execution Time Analysis, 2012

[19] **Plateau, B.**: "*On the Stochastic Structure of Parallelism and Synchronization Models for Distributed Algorithms*", ACM SIGMETRICS, Vol. 13, pp. 147-154, 1985

[20] **Ciardo, G.** and **Miner, A. S.**: "*A Data Structure for the Efficient Kronecker Solution of GSPNs*", in Proc. 8th Int. Workshop on Petri Nets and Performance Models (PNPM'99), 1999

[21] **Cui, Y.**: *Simulation-Based Hybrid Model for a Partially-Automatic Dispatching of Railway Operation*, I. f. E. u. Verkehrswesen, Ed., Universität Stuttgart: PhD thesis, 2010

[22] **Martin, U.**: *Verfahren zur Bewertung von Zug- und Rangierfahrten bei der Disposition*, TU Braunschweig: PhD thesis, 1995

[23] **Pachl, J.**: *Steuerlogik für Zuglenkanlagen zum Einsatz unter stochastischen Betriebsbedingungen*, TU Braunschweig: PhD thesis, 1993

[24] **Petersen, E.** and **Taylor, A.**: "*Line Block Prevention in Rail Line Dispatch*", INFOR Journal, Vol. 21, No. 1, pp. 46-51, 1983

[25] **Mills, G.**, **Pudney, P. J.**, **White, K.** and **Hewitt, J.**: "*The effects of deadlock avoidance on rail network capacity and performance*", in Proc. of the 2003 mathematics-in-industry study group, 2003

[26] **Lu, Q.**, **Dessouky, M.** and **Leachman, R. C.**: "*Modeling Train Movements Through Complex Rail Networks*", ACM Transactions on Modeling and Computer Simulation, Vol. 14, No. 1, pp. 48-75, 2004

[27] **Fanti, M. P.**, **Giua, A.** and **Seatzu, C.**: "*A deadlock prevention method for railway networks using monitors for colored Petri nets*", in Proc. of Int. Conf. on Systems, Man and Cybernetics, 2003

[28] **Zarnay, M.**: "*Solving deadlock states in model of railway station operation using coloured Petri nets*", in Proceedings of Symposium FORMS/FORMAT, 2008

[29] **Zehfuss, J. G.**: "*Ueber eine gewisse Determinante*," Zeitschrift für Mathematik und Physik, Vol. 3, pp. 298-301, 1858