, , 1–33 ()

Verification of safety properties using integer programming: Beyond the state equation*

JAVIER ESPARZA AND STEPHAN MELZER {esparza, melzers}@informatik.tu-muenchen.de
Institut für Informatik, Technische Universität München
Arcisstr. 21, 80290 München, Germany

Editor: Joseph Sifakis

Abstract. The state equation is a verification technique that has been applied - not always under this name – to numerous systems modelled as Petri nets or communicating automata. Given a safety property $\mathcal{P}$, the state equation is used to derive a necessary condition for $\mathcal{P}$ to hold which can be mechanically checked. The necessary conditions derived from the state equation are known to be of little use for systems communicating by means of shared variables, in the sense that many of these systems satisfying the property but not the conditions. In this paper, we use *traps*, a well-known notion of net theory, to obtain stronger conditions that can still be efficiently checked. We show that the new conditions significantly extend the range of verifiable systems.

Keywords: State equation, traps, approximation techniques, linear programming

1. Introduction

The application of linear algebra and integer programming techniques to verification problems has been the subject of a large number of papers [3, 4, 8, 10, 11, 13, 22, 34]. One of the most popular techniques is what can be called the *state equation* [34], due to its similarity with the concept of state equation in control theory. Loosely speaking, the state equation is a linear approximation to the system's behaviour that can be derived (in linear time) from the syntactic descriptions of the system and its initial state. More formally, it is a linear matrix equation, or equivalently a set of linear constraints L, that every reachable state must satisfy. In other words, the solutions of L are a superset of the reachable states.

The state equation can be applied to verification problems in the following way. If the states of the system that do *not* satisfy a property $\mathcal{P}$ can be encoded as the solutions of a set of linear constraints $L_{\neg\mathcal{P}}$, then one can construct the set of constraints $L \cup L_{\neg\mathcal{P}}$, and use well-known algorithms of mathematical programming to check if it has some solution. If there is no solution, then $\mathcal{P}$ holds for every reachable state.

The disadvantage of this technique is the fact that the states satisfying L are only a superset of the reachable markings: the solutions of $L \cup L_{\neg\mathcal{P}}$ may or may not correspond to a reachable state. Therefore, the state equation only leads to a *positive test* for the property $\mathcal{P}$: If the system passes the test then it satisfies $\mathcal{P}$; if it fails the test it may satisfy $\mathcal{P}$ or not.[1] The main advantages of the state equation

* This work has been partially supported by the Sonderforschungsbereich SFB 342 A3 SAM. Some of the results appeared in the proceedings of ESOP '96 [29].

are that it does not explore the state space, and so it avoids the state explosion problem, and that it can also be used to verify systems having infinite state spaces.

The state equation technique can be applied to different models of concurrency. Although it has long been applied to Petri net models [8, 34], the most comprehensive study of its applications for verification has been carried out by Avrunin, Corbett *et al.* on systems modelled as coupled automata [3, 4, 10, 11].[2] In particular, these authors have developed the Constrained Expression Toolset, later updated to the Inequality Necessary Condition Analyzer (INCA), a tool for the verification of a large class of safety and liveness properties. In [10], Corbett shows that INCA is able to prove deadlock freedom for a number of systems modelled in ADA, and can compete with symbolic and partial order model checkers.

One of the main limitations of the state equation is that it tends to fail for systems which communicate via shared variables. For instance, it cannot prove mutual exclusion of any of the most popular mutual exclusion algorithms (Dekker's, Dijkstra's, Knuth's, Peterson's etc.) without user's help. The reason is that the method is not sensitive to the guards which allow to perform an action only if a variable has a certain value; more precisely, the state equation for the system with the guards coincides with the state equation for the system without the guards (or with all guards set to true). Since the correctness of these algorithms crucially depends on these guards, the method fails.

In this paper, we show how to sharpen the state equation test by means of *traps*. Traps are sets of *local* states (states of a component of the system) which can be easily computed from the syntactic description of the system, and have the following property: If a global state g contains some local state of a trap, then so does every successor of g (though not necessarily the same local state). Each trap leads to a constraint which improves the approximation of the state equation to the set of reachable states, and is sensitive to the presence or absence of guards. We define a new test, called the *trap test*, which uses the information contained in these constraints.

The practical interest of any positive test for a property $\mathcal{P}$ is directly proportional to its *quality* (how often systems satisfying $\mathcal{P}$ pass the test) and inversely proportional to its computational *cost* (the complexity of deciding if a system passes the test or not). We show that the trap test has the same computational complexity as the state equation test (both are NP-complete) and a significantly better quality. This we do by exhibiting a set of examples (five mutex algorithms, two telephone protocols and a production cell) which can be verified with the trap test, but not with the state equation test.

Once a test has been designed, it is necessary to provide algorithms that *perform* it. Even if two tests have the same cost, the algorithms that perform one of them may profit from good heuristics which may not be applicable to the second. The state equation test is an integer or mixed integer programming problem, depending on which variant is considered, and so it profits from all the heuristics developed for solving these problems. For the trap test we obtain two results. First, we prove that it can also be reduced to a mixed integer programming problem. Unfortunately, this approach turns out to be rather unsuitable when applied to large systems. So,

second, we show how to perform the test by means of an algorithm which iteratively adds to the constraints of the state equation new constraints derived from traps.
The paper is structured as follows. Section 2 contains a few basic definitions on Petri nets and mathematical programming. Section 3 introduces the state equation test, and discusses briefly its quality and cost. Section 4 does the same for the trap test. In Section 5 the trap test is reduced to a mixed integer programming problem, and the pros and cons of this approach are discussed. Section 6 introduces the iterative algorithm to perform the trap test. Section 7 is devoted to experimental results.

2. Preliminaries

Petri nets. A *net* is a triple $N = (P, T, F)$ where $P \cap T = \emptyset$ and $F \subseteq (P \times T) \cup (T \times P)$. P is the set of *places*, T the set of *transitions* and F is the *flow relation*. We identify F with its characteristic function $(P \times T) \cup (T \times P) \to \{0, 1\}$. The *pre-set* of $x \in P \cup T$ is ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$. The *post-set* of $x \in P \cup T$ is $x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}$. The pre- and post-set of a subset of $P \cup T$ are the union of the pre- and post-sets of its elements.
A *marking* of N is a function $M: P \to \mathbb{N}$, which describes the number of *tokens* $M(p)$ that the marking puts in each place p. In the paper we represent markings as multisets of places; for instance, the multiset $\{p_1, 2p_2\}$ represents the marking that puts one token on p_1, two tokens on p_2, and no tokens elsewhere.
A *Petri net* is a pair (N, M_0) where N is a net and M_0 a marking of N called *initial marking*. Petri nets are graphically represented as follows: places and transitions are represented as circles and boxes, respectively. An element (x, y) of the flow relation is represented by an arc leading from x to y. A token on a place p is represented by a black dot in the circle corresponding to p.
A transition $t \in T$ is *enabled at* M iff $\forall p \in {}^\bullet t : M(p) \geq F(p, t)$. If t is enabled at M, then t may *fire* or *occur*, yielding a new marking M' (denoted by $M \xrightarrow{t} M'$), where $M'(p) = M(p) + F(t, p) - F(p, t)$.
A sequence of transitions, $\sigma = t_1 t_2 \ldots t_r$ is an *occurrence sequence* of (N, M_0) iff there exist markings $M_1, \ldots M_r$ such that $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \ldots \xrightarrow{t_r} M_r$. The marking M_r is said to be *reachable* from M_0 by the occurrence of σ (denoted $M \xrightarrow{\sigma} M_r$).
A Petri net (N, M_0) is *1-safe* iff $M(p) \leq 1$ for every reachable marking M and every place p.
Linear, Integer, and Mixed Integer Programming. A *linear programming problem* is a set of linear (in)equations or *constraints* on a given set of variables, plus maybe a linear function on the same variables called the *objective function*. Using vector and matrix notation, a linear programming problem with objective function takes the form

$$\begin{array}{l} \text{Variables: } X \\ \text{Maximise } {}^t c \cdot X \text{ subject to:} \\ A \cdot X \ \leq \ b \end{array}$$

where A is a matrix, and b, c are columnn vectors.
A solution of the problem is a vector of rational numbers that satisfy the constraints. A solution is *optimal* if it maximizes the value of the objective function (over the set of all solutions).
An *integer programming problem* consists of the same elements as a linear programming problem, but only integer solutions are allowed. In a *mixed integer programming problem* (shortened to *mixed programming problem* in the sequel) some variables may take rational values, while others must take integer ones.
A linear, integer or mixed programming problem is *feasible* if it has a solution. Otherwise it is *infeasible*.
It is well known that feasability of linear programming problems can be solved in polynomial time, while feasability of mixed or integer programming problems is NP-complete [39].

3. The state equation test

In this section we briefly introduce the state equation in a Petri net setting, and determine the computational cost of the state equation test.
Given an occurrence sequence $M_0 \xrightarrow{\sigma} M$ of a Petri net (N, M_0), we easily see that each place p satisfies a "token conservation law": the number of tokens that M puts in p is equal to the number of tokens that M_0 puts in p *plus* the number of tokens added by the occurrence of p's input transitions *minus* the number of tokens removed by the occurrence of p's output transitions. If we let $\#(\sigma, t)$ denote the number of times that t occurs in σ, then the token conservation law is expressed by the following equation:

$$M(p) = M_0(p) + \sum_{t \in {}^\bullet p} \#(\sigma, t) - \sum_{t \in p^\bullet} \#(\sigma, t).$$

The system of equations representing the token conservation law for all places can be written in matrix form as

$$M = M_0 + C \cdot \vec{\sigma}$$

where C denotes the *incidence matrix* of N, a $P \times T$ integer matrix given by

$$C(p, t) = F(t, p) - F(p, t)$$

and $\vec{\sigma}$ is the *Parikh vector* of σ, given by $\vec{\sigma} = (\#(\sigma, t_1), \ldots, \#(\sigma, t_n))$.
Thus, for a given Petri net and a given marking M, we can formulate the following linear programming problem, called the *state equation*:

$$\begin{array}{l} \text{Variables: } X \\ M \;=\; M_0 + C \cdot X \\ X \;\geq\; \vec{0} \end{array}$$

For every marking M reachable from M_0, i.e., for every marking such that $M_0 \xrightarrow{\sigma} M$ holds for some σ, the state equation has at least one solution, namely $X := \vec{\sigma}$. So the feasibility of the state equation is a necessary condition for the reachability of the marking M.
We wish to verify that every reachable marking satisfies a desirable property, or, equivalently, that no marking satisfying the negation of this desirable property is reachable. The negation of the property can often be expressed by means of *linear constraints* on the markings of the net. Here are two examples:

- Mutual exclusion.
  In Petri net models of mutual exclusion algorithms the possible states of a process (idle, requesting, critical, ...) are modeled by places which can hold at most one token. The process is in the critical section if the corresponding place is marked. If $s_1, \ldots, s_n$ are the places corresponding to the critical sections, then the reachable markings that violate the mutual exclusion property are those satisfying
  $$M(s_1) + \ldots + M(s_n) \geq 2$$
- Deadlock freedom in 1-safe Petri nets.
  A marking is a deadlock if it does not enable any transition. In 1-safe Petri nets, where a place can hold at most one token, a transition with k input places is enabled at a marking M if M puts at least k tokens in its input places. In other words, the deadlocked markings satisfy
  $$\sum_{p \in {}^\bullet t} M(p) \leq |{}^\bullet t| - 1$$
  for every transition t.

A *linear property* of a net N is a predicate $\mathcal{P}$ on the markings of N such that

$$\mathcal{P}(M) \Leftrightarrow A \cdot M \geq b$$

for some matrix A and vector b.[3] A property is *co-linear* if its negation is linear. If some marking M_1 satisfying a linear property $\mathcal{P}$ is reachable from M_0 by means of a sequence σ, then the *generalised state equation*

$$\begin{array}{rcl} \text{Variables: } M,\ X & & \\ M & = & M_0 + C \cdot X \\ A \cdot M & \geq & b \\ M, X & \geq & \vec{0} \end{array}$$

has an integer solution $X := \vec{\sigma}, M := M_1$. Therefore, if the generalized state equation is infeasible, then every reachable marking satisfies the negation of $\mathcal{P}$.
So we have the

> *State equation test* (for a co-linear property): check if the generalised state equation is feasible. The system passes the test if the equation has no solution, and fails it otherwise.

Actually, we have not completely defined the test yet, because we have to specify the *domain* in which the equation should be feasible. We have three variants of the state equation test:

(1) both M and X must be integer (the integer test);

(2) M must be integer, X can be rational (the mixed test);

(3) both M and X can be rational (the rational test).

Observe that the fourth variant, M can be rational but X must be integer, is excluded because the matrix C is integer valued, and so if X is integer then automatically so is M.
We study the ratio quality/cost for the rational, integer, and mixed tests, in this order.

3.1. The rational test

Since linear programming problems can be solved in polynomial time, the rational test has polynomial time cost. Unfortunately, the experiments show that this test has very poor quality: in most systems the generalised state equation has rational solutions even if the co-linear property holds. None of the examples studied by Corbett in [10] and none of the examples considered in this paper (making a total of 27 models) can be verified using the rational test. For this reason we do not consider the rational test any further.

3.2. The integer test.

The integer test is essentially the one used in [10, 11]. It has much better quality. It allows to prove deadlock freedom for the examples of [10], and for five of the six mutex algorithms studied in this paper (see Section 7). Moreover, it is an exact algorithm (i.e., a co-linear property holds if *and only if* the system passes the test) for acyclic Petri nets [34][4].
The price to pay is a higher cost:

THEOREM 1 *The problem of deciding if a Petri net fails an integer state equation test is NP-complete, even for the case in which A is an identity matrix.*

Proof: We prove NP-completeness of the problem of deciding if the generalised state equation has a solution M, X with both M and X integer. Since feasability of integer programming problems is in NP, so is our problem. In the case of arbitrary A, NP-hardness follows immediately from the NP-hardness of determining if $A \cdot M \geq b$ alone has an integer solution. The NP-hardness of the case in which A is an identity matrix can be proved in several ways; here we give a proof based on the result of net theory mentioned above: for acyclic Petri nets, a marking M is reachable if and only if the state equation $M = M_0 + C \cdot X$ has an integer solution X [34].

We proceed by reduction from the satisfiability problem for boolean formulae in conjunctive normal form [19]. Given an instance ϕ of this problem with n variables $x_1, \ldots x_n$ and m clauses $c_1, \ldots c_m$, we construct the Petri net (N_ϕ, M_{0_ϕ}), and the vector b_ϕ in the following way (recall that A_ϕ is the identity matrix). N_ϕ contains:

- a place p_{x_i} for each variable x_i; the preset of p_{x_i} is empty, and its postset contains two transitions t_{x_i} and $t_{\overline{x_i}}$;
- a place $p_{j,l}$ for each clause c_j and each literal l of c_j; the preset of $p_{j,l}$ is the transition x_l; its postset is a transition $t_{j,l}$.
- a place p_{c_j} for each clause c_j; the preset of p_{c_j} contains the transition $t_{j,l}$ for each literal l of c_j; its postset is empty.

M_{0_ϕ} puts one token on all places p_{x_i}, and no tokens elsewhere. The vector b_ϕ is given by $b_\phi(p) = 1$ if p is one of the places p_{c_j}, and 0 otherwise.
Clearly, ϕ is satisfiable if and only if (N_ϕ, M_{0_ϕ}) has a reachable marking which puts at least one token on each place p_{c_j}. Since the net N_ϕ is acyclic, this is the case if and only if the generalised marking equation with A_ϕ the identity matrix and b_ϕ as described above has an integer solution. ■

3.3. The mixed test

Maybe surprisingly, the mixed and integer tests have very similar quality and cost, and experimental results show that they perform similarly in practice.
Quality: Although it is easy to construct systems that can be verified with the integer test but not with the mixed test, none of the examples we have examined so far (including those of [10]) exhibits this property.
Cost: The mixed test is NP-complete. To prove this, we show that the integer test for a Petri net (N, M_0) and a property can be polynomially reduced to the mixed test for another Petri net (N', M_0') and the same property. The net N' is obtained from N by adding for each transition t a new place p_t satisfying ${}^\bullet p_t = \{t\}$ and $p_t^\bullet = \emptyset$. M_0' is defined as the marking that puts as many tokens as M_0 in the places of N, and no tokens in the new places. In order to see that (N, M_0) passes the integer test if and only if (N', M_0') passes the mixed test, observe that for each of the new places p_t of N' the state equation has the form $M(p_t) = X(t)$. Therefore, there can exist no solution of the generalised state equation with M integer and X non-integer.
Experimental results: Even though the two tests have the same cost, one would expect the mixed test to perform better, since the number of integer variables is considerably smaller. This is not the case if the integer test is implemented with care.[5] The times taken by the mixed and integer test for deadlock-freedom in the examples of [10] are then very similar.
In Section 6 we introduce an algorithm to perform the trap test, for which there is also a mixed and an integer version. In this case we will observe a small difference in performance in favour of the mixed test.

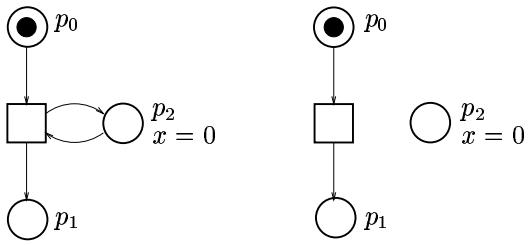

Figure 1. A limitation of the state equation

3.4. A limitation of the state equation test

Unfortunately, the quality of both the mixed and the integer test is still poor when applied to distributed systems communicating through shared variables. One of the reasons is that the components of such systems have to test the value of a variable in order to determine the flow of control. Now, consider the two Petri nets of Figure 1. The Petri net on the left models a component which may change its local state from p_0 to p_1 only if the variable x has value 0, which happens not to be the case. In the Petri net on the right, the component can change its local state independently of the value of x. Obviously, the marking $\{p_1\}$ is not reachable on the left, and reachable on the right. However, the state equations of these two nets *coincide*. They are in both cases:

$$\begin{array}{lcl} M(p_0) & = & 1 - X(t) \\ M(p_2) & = & 0 + X(t) - X(t) \;=\; 0 \\ M(p_1) & = & 0 + X(t) \qquad\quad = \; X(t) \end{array}$$

Therefore, the generalised state equation cannot be used to prove that $\{p_1\}$ is not reachable on the left.

As a more interesting example, consider the Petri net of Figure 2, which models Lamport's 1-bit mutex algorithm for two processes.

Two arcs connecting a transition and a place in both directions are graphically represented as an arc with two arrows. The critical sections of the two processes are modelled by the places cs_1 and cs_2.

The system fails the state equation test for the mutual exclusion property, because the marking $\{cs_1, nid_1, cs_2\}$ is (the unique) solution of the corresponding generalised state equation.

We could of course prove these properties by constructing the reachability graph, which is very small in these examples. However, in larger examples the number of reachable states may grow exponentially in the size of the net, or even be infinite. In the next section we present an alternative technique which refines the state equation test by means of traps.

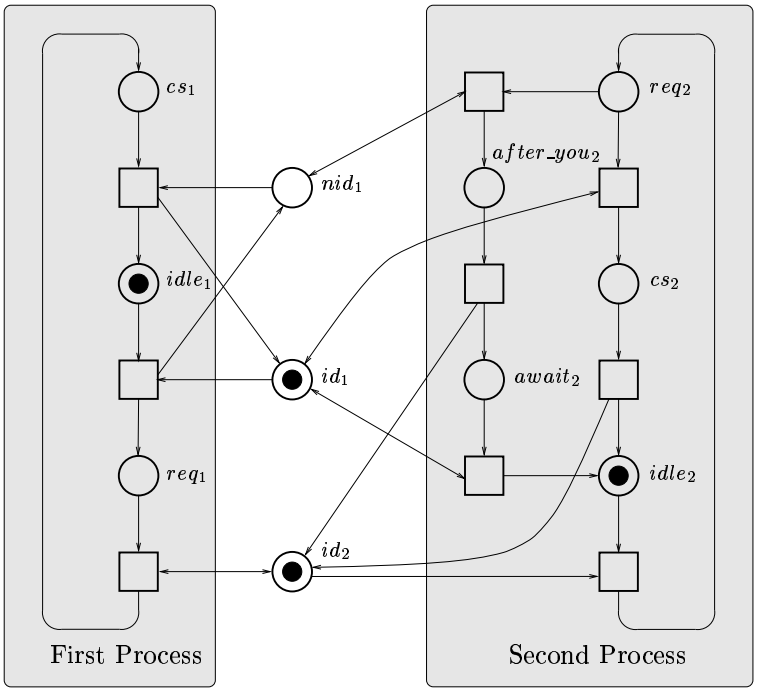

Figure 2. A Petri net model of Lamport's 1-bit algorithm

4. The trap test

A *trap* Θ of a net is a set of places such that every transition in the postset of Θ is included in the preset of Θ, i.e., $\Theta^\bullet \subseteq {}^\bullet\Theta$ [38, 15]. Traps have the following fundamental property:

PROPOSITION 1 *Let (N, M_0) be a Petri net, where $N = (P, T, F)$, and let $\Theta \subseteq P$ be a trap of N. If Θ is initially marked, i.e., $\sum_{p \in \Theta} M_0(p) > 0$, then it remains marked at every marking reachable from M_0.*

Proof: It suffices to prove that if a trap Θ is marked at a marking M_1, and $M_1 \xrightarrow{t} M_2$ for some transition t, then Θ remains marked at M_2. If $t^\bullet$ contains some place of Θ, then its occurrence puts tokens in Θ, and so Θ is marked at M_2. Otherwise, since $\Theta^\bullet \subseteq {}^\bullet\Theta$, the occurrence of t neither removes nor puts tokens in Θ, and so, since Θ was marked at M_1, it is marked at M_2. ■

Using a trap we can easily show that the marking $\{p_1\}$ is not reachable in the net on the left of Figure 1. The set $\{p_0, p_2\}$ is an initially marked trap of the net on the left. However, the trap is not marked at the marking $\{p_1\}$. Therefore, $\{p_1\}$ is not reachable. We can also prove mutual exclusion for Lamport's algorithm: the marking $\{cs_1, nid_1, cs_2\}$ of the net of Figure 2, the only solution of the generalised state equation, does not mark the trap $\{req_1, id_1, id_2, req_2, after\_you_2\}$, which is initially marked. Therefore, this marking is not reachable.
If a marking marks every trap that is marked at M_0 we say that it satisfies the *trap property*. It follows from Proposition 1 that every reachable marking satisfies not only the state equation, but also the trap property. We have thus a refined test for co-linear properties, which we call the

> *Trap test* (for a co-linear property): check if the generalised state equation has a solution M, X such that M satisfies the trap property. The system passes the test if there is no such solution, and fails it otherwise.

We have again the rational, integer, and mixed variants of the test. The rational variant faces the same problems as the rational state equation test, and so we do not consider it any further.
We show that the computational cost of the integer trap test coincides with that of the integer state equation test:

THEOREM 2 *The problem of deciding if a Petri net fails an integer trap test is NP-complete, even if A is a diagonal matrix.*

Proof: We prove that the problem of deciding if the generalised state equation has an integer solution M, X such that M satisfies the trap property is NP-complete. NP-hardness is proved as for the state equation test. Membership in NP is proved by the following nondeterministic polynomial time algorithm:

1. Guess a set $Q \subseteq P$ of places of N. (This will be the set of places marked at M.)

2. Check that every trap marked at M_0 contains some place of Q. (A polynomial algorithm for this problem can be found in [40, 15], and is also briefly described in the next section.)

3. For each place p, if $p \in Q$ then add $M(p) \geq 1$ to the generalised state equation; otherwise add $M(p) \leq 0$.

4. Check in nondeterministic polynomial time that the resulting equation system has a solution M, X with M integer.

The deterministic algorithm for the trap test that can be derived from the proof of this result is very inefficient, since it requires to enumerate all subsets of places of P, or at least all the subsets that are not included in each other. In sections 5 and 6 we investigate possible alternatives.

4.1. A limitation of the trap test

Since the reachability problem is PSPACE-hard even for 1-safe Petri nets [9] but the trap test has only NP-complete cost, one should not expect the test to have perfect quality. Figure 3 shows an example of a net in which the trap test fails to prove a property, namely mutual exclusion of the places p_3 and p_5. The net fails the trap test because

- $M = \{p_3, p_5\}$ and $X(t_1) = X(t_2) = X(t_3) = 1$ are a solution of the generalised state equation (the state equation plus the constraint $M(p_3) + M(p_5) \geq 2$), and
- M marks all traps marked at M_0: Every trap containing p_1 contains also p_5 or p_3, and every trap containing p_4 contains also p_5.

Despite this limitation the trap test turns out to have interesting applications, as shown in Section 7.

5. The trap inequation

In this section we show that the trap test (both its integer and mixed variants) can be reduced to a mixed programming problem. More precisely, we show that whether a marking satisfies the trap property or not can be reduced to the problem of deciding if a *trap inequation*, very similar to the state equation, has some solution. We can then put the generalised state equation and the trap inequation together to yield an augmented system of (in)equations; a system passes the trap test if and only if the system has no solution.

5.1. Linear algebraic representation of traps

We start with an idea due to Alaiwan and Toudic [1]: Given a Petri net (N, M_0) where $N = (P, T, F)$, we construct a system of linear inequations over the variables

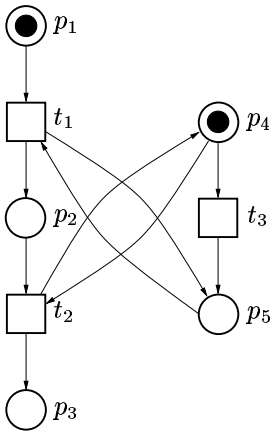

Figure 3. A net showing a limitation of the trap test.

$\{y_p \mid p \in P\}$ satisfying the following property: a set $Q \subseteq P$ is a trap if and only if the system of inequations has a rational solution $\{y_p := a_p \mid p \in P\}$ such that for every place p:

$$a_p > 0 \text{ if } p \in Q \quad \text{and} \quad a_p = 0 \text{ if } p \notin Q.$$

Recall the definition of trap (slightly reformulated): Q is a trap if for every place $p \in Q$ and every transition $t \in p^\bullet$, some place of $t^\bullet$ belongs to Q. This conditional is translated into the inequation $y_p \leq \sum_{q \in t^\bullet} y_q$. So the system of inequations contains:

$$\begin{array}{lll} y_p & \leq & \displaystyle\sum_{q \in t^\bullet} y_q \quad \text{for each place } p \text{ and transition } t \in p^\bullet \\ y_p & \geq & 0 \qquad\qquad\quad \text{for each place } p \end{array}$$

Observe that there are $O(|P| \cdot |T|)$ inequations over $|P|$ variables.
If Q is a trap, then the assignment

$$y_p := \begin{cases} 1 & \text{if } p \in Q \\ 0 & \text{otherwise} \end{cases}$$

is a solution of the system. Conversely, if $\{y_p := a_p \mid p \in P\}$ is a solution, then for every place p such that $a_p > 0$ and every transition $t \in p^\bullet$ we have $\sum_{q \in t^\bullet} a_q > 0$, and so $a_q > 0$ for some $q \in t^\bullet$.
The inequations corresponding to the Petri net of Figure 4 are (where we shorten y_{p_i} to y_i):

$$\begin{array}{lll} y_1 \leq y_3 & & y_4 \leq y_5 + y_6 \\ y_2 \leq y_4 & & y_5 \leq y_3 \\ y_3 \leq y_5 + y_6 & & y_6 \leq y_4 \\ y_1, \ldots, y_6 \geq 0 & & \end{array}$$

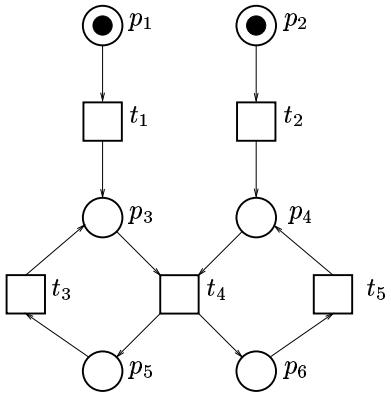

Figure 4. A small Petri net.

Let $Y = (y_1, \ldots y_6)$. The assignments $Y := (0,0,1,0,1,0)$ and $Y := (0,1,1,1,1,0)$ are solutions of the inequations above, and so $\{p_3, p_5\}$ and $\{p_2, p_3, p_4, p_5\}$ are traps of the net.
It is convenient to write the system of inequations in matrix form. Let C_Θ be a $P \times (F \cap (P \times T))$-matrix defined in the following way:

$$C_\Theta(p,(q,t)) = \begin{cases} -1 & p = q \wedge p \notin t^\bullet \\ 1 & p \neq q \wedge p \in t^\bullet \\ 0 & \text{otherwise} \end{cases}$$

Then, the system takes the form

$$\begin{array}{rl} & \text{Variables: } Y \\ {}^tY \cdot C_\Theta & \geq \vec{0} \\ Y & \geq \vec{0} \end{array}$$

which we call the linear algebraic representation of traps.

5.2. The trap inequation

We construct for a given marking M a *trap inequation* that has a rational solution if and only if M satisfies the trap property. We proceed in three steps:

(1) Using the linear algebraic characterization of traps, we derive a system of inequations that has a rational solution if and only if M *violates* the trap property; we call this system the *primal* system.

(2) We construct the so-called dual system of inequations, and with the help of (a variant of) the Minkowski-Farkas lemma we show that it has a rational solution if and only if M *satisfies* the trap property.

(3) The result of putting together the dual system and the generalised state equation is a non-linear system; we fix this problem by massaging the dual system, and arrive so at the trap inequation.

Here is the primal system:

$$\begin{array}{rcll} \text{Variables: } Y & & & \\ {}^tY \cdot C_\Theta & \geq & \vec{0} & (Y \text{ corresponds to a trap } Q_Y) \\ Y & \geq & \vec{0} & \\ {}^tY \cdot M & = & 0 & (Q_Y \text{ is unmarked at } M) \\ {}^tY \cdot M_0 & > & 0 & (Q_Y \text{ is marked at } M_0) \end{array}$$

Clearly, it has a solution if and only if there is a trap marked at M_0 but not at M, and so if and only if M violates the trap property.
The Minkowski-Farkas lemmata (see for instance [39]) state that given a system of linear equations or inequations of a certain form, called the primal system, there is a dual system which is feasible (has a rational solution) if and only if the primal system is infeasible (has no rational solution). The following lemma belongs to the group:

LEMMA 1 (MINKOWSKI-FARKAS) *One and only one of the following two problems is feasible:*

$$\begin{array}{rcl} \textit{Variables: } X & & \\ A \cdot X & \leq & b \\ X & \geq & \vec{0} \end{array} \qquad \begin{array}{rcl} \textit{Variables: } Y & & \\ {}^tY \cdot A & \geq & \vec{0} \\ {}^tY \cdot b & < & 0 \\ Y & \geq & \vec{0} \end{array}$$

In order to apply this result, we first manipulate the primal system. We observe that, since M is a nonnegative vector and any solution Y must also be nonnegative, the constraint ${}^tY \cdot M = 0$ can be safely replaced by ${}^tY \cdot M \leq 0$. So the primal system is equivalent to (i.e., has the same solutions as):

$$\begin{array}{rcl} \text{Variables: } Y,\ M & & \\ {}^tY \cdot (C_\Theta | -M) & \geq & \vec{0} \\ {}^tY \cdot (-M_0) & < & 0 \\ Y & \geq & \vec{0} \end{array}$$

where $(C_\Theta | -M)$ denotes the matrix obtained by adding $-M$ to C_Θ as rightmost column.
By Lemma 1, the dual system

$$\begin{array}{rcl} \text{Variables: } Z & & \\ (C_\Theta \mid -M) \cdot Z & \leq & -M_0 \\ Z & \geq & \vec{0} \end{array}$$

is feasible if and only if the primal system is unfeasible, and so if and only if M satisfies the trap property.
We are not ready yet. If we put the dual system together with the generalised trap equation we obtain non-linear inequations, since now both M and Z are variables. To solve this problem we massage the dual system.
Notice that the dimension of Z is equal to the number of arcs leading from places to transitions plus 1, because of the addition of the column M. Define $Z = (Z' \mid z')$, i.e., Z' is the vector containing all the components of Z but the last, and z' is the last component of Z. With these notations, we can rewrite the dual system as:

$$\begin{array}{rcl} \text{Variables: } Z',\ z' & & \\ z'M & \geq & M_0 + C_\Theta \cdot Z' \\ Z' & \geq & \vec{0} \\ z' & \geq & 0 \end{array}$$

If the system has a solution with $z' = 0$, then, since M is nonnegative, it also has a solution for every $z' > 0$. So we can safely replace $z' \geq 0$ by $z' > 0$, and the resulting system is still feasible if and only if M satisfies the trap property. Now, since $z' > 0$, we can divide the first inequality by it. Redefining $Z := \frac{1}{z'}Z'$ and $z := \frac{1}{z'}$, we finally get the *trap inequation*

$$\begin{array}{rcl} \text{Variables: } Z,\ z & & \\ M & \geq & zM_0 + C_\Theta \cdot Z \\ Z & \geq & \vec{0} \\ z & > & 0 \end{array}$$

whose name is due to its similarity with the state equation.
This derivation of the trap inequation is indirect, but unfortunately we have not found any simple argument that could replace the application of Lemma 1 (we have found rather complicated arguments which are more or less equivalent to proving the lemma). It is easy to directly prove that if the equation has a solution Z_0, z_0, then every trap marked at M is marked at M_0. For that, let $Y_Q \geq 0$ be the linear algebraic representation of a trap Q marked at M_0. We have

$$^tY_Q \cdot M \geq z_0 {}^tY_Q \cdot M_0 + {}^tY_Q \cdot C_\Theta \cdot Z_0$$

Since $^tY_Q \cdot C_\Theta \geq 0$ and $^tY_Q \cdot M_0 > 0$, we have $^tY_Q \cdot M > 0$, which implies that Q is marked at M. The difficult part is to prove that if every trap marked at M is marked at M_0 then the equation has a solution; for this part, Lemma 1 or an equivalent result seems to be necessary.
The *generalised trap inequation* is the result of putting together the generalised trap equation and the trap inequation. It can be solved using mixed programming techniques. The variables Z, z are always rational, the variable M is always integer, and X can be required to be either rational or integer, according to whether we wish to perform the mixed or the integer variant of the test.
Observe that mixed programming solves inequation systems of the form $A \cdot X \leq b$. Since the constraint $z > 0$ does not fit in this format, we solve an optimization

problem: we solve the system with $z \geq 0$ as constraint, but search for the solution with maximal value of z.

$$\begin{array}{rcl} \text{Variables: } M,\ X,\ Z,\ z & & \\ \text{Maximise } z \text{ subject to;} & & \\ M & = & M_0 + C \cdot X \\ M & \geq & zM_0 + C_\Theta \cdot Z \\ A \cdot M & \geq & b \\ M, X, Z & \geq & \vec{0} \\ z & \geq & 0 \end{array}$$

The system passes the test if and only if $z_{op} > 0$, where z_{op} is the value of z in the optimal solution.

5.3. The numerical limit of the trap inequation

Most packages for mathematical programming work with a certain precision. Assume that a particular package has precision ϵ, and assume further that the generalised trap equation has an optimal solution $0 < z_{op} < \epsilon$. Since the package cannot distinguish 0 from z_{op}, it reports $z_{op} = 0$. So the system fails the trap test, but the package reports that it passes it.
For the results to be reliable, we have to restrict ourselves to systems for which $z_{op} > 0$ implies $z_{op} \geq \epsilon$. Unfortunately, there are even small systems which do not satisfy this condition:

PROPOSITION 2 *For every $n \geq 1$ there is a Petri net (N_n, M_{0n}) with $n+1$ transitions and $n+2$ places such that the optimal solution of*

$$\begin{array}{rcl} \textit{Variables: } M,\ X,\ Z,\ z & & \\ \textit{Maximise } z \textit{ subject to:} & & \\ M & = & M_0 + C^n \cdot X \\ M & \geq & zM_0 + C^n_\Theta \cdot Z \\ M(p_1) & = & 0 \\ & \cdots & \\ M(p_{n+1}) & = & 0 \\ M(p_{n+2}) & = & 1 \\ M, X, Z & \geq & \vec{0} \\ z & \geq & 0 \end{array}$$

where C^n and C^n_Θ are the matrices corresponding to N_n, satisfies $0 < z_{op} \leq 2^{-n}$.

Proof: Let (N_n, M_{0n}) be as shown in Figure 5.
Recall that the vector Z has a component for each arc (p,t) leading from a place p to a transition t; we call this component $Z(p,t)$. It is easy to see that the system

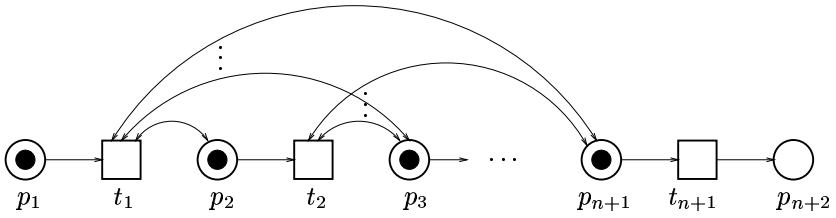

Figure 5. Petri net N_n with $z_{op} \leq 2^{-n}$.

$M \geq zM_0 + C_{\ominus}^n \cdot Z$ looks as follows:

$$\begin{array}{rcl}
M(p_1) = 0 & \geq & z - Z(p_1,t_1) \\
M(p_2) = 0 & \geq & z - Z(p_2,t_2) + Z(p_1,t_1) + c_1 \\
M(p_3) = 0 & \geq & z - Z(p_3,t_3) + Z(p_2,t_2) + Z(p_1,t_1) + c_2 \\
\cdots & & \\
M(p_{n+1}) = 0 & \geq & z - Z(p_{n+1},t_{n+1}) + Z(p_n,t_n) + \ldots + Z(p_1,t_1) + c_n \\
M(p_{n+2}) = 1 & \geq & Z(p_{n+1},t_{n+1})
\end{array}$$

where

$$c_i = \sum_{\substack{k \;=\; 1,\ldots i \\ j \;=\; k+1,\ldots,n+1}} Z(p_j,t_k)$$

Since $Z \geq 0$, we have $c_i \geq 0$ for every i. This implies

$$\begin{array}{rcll}
Z(p_1,t_1) & \geq & z & \\
Z(p_2,t_2) & \geq & z + Z(p_1,t_1) & \geq 2z \\
Z(p_3,t_3) & \geq & z + Z(p_2,t_2) + Z(p_1,t_1) & \geq 4z \\
 & \cdots & & \\
Z(p_{n+1},t_{n+1}) & \geq & z + Z(p_n,t_n) + \ldots + Z(p_1,t_1) & \geq 2^n z \\
1 & \geq & Z(p_{n+1},t_{n+1}) &
\end{array}$$

and so $z \leq 2^{-n}$ for any solution z. In particular, we have $z_{op} \leq 2^{-n}$.
On the other hand, it is easy to see that $X := (1,1,\ldots,1)$, $M := (0,0,\ldots,1)$,

$$Z(p_i,t_j) := \begin{cases} 2^{n+1-i} & \text{if } i = j \\ 0 & \text{otherwise} , \end{cases}$$

and $z := 2^{-n}$ are a solution of the system. So we have $z_{op} \geq 2^{-n}$, and so in particular $z_{op} > 0$. ■

Due to this result, a package with precision ϵ cannot be safely used to solve the generalised trap equation for Petri nets having more than $-log_2\epsilon + 2$ places. In the case of CPLEX^{TM} (version 3.0), the package we have used in our experiments,

$\epsilon = 10^{-6}$, and so the upper limit in the number of places is about 20, which is insufficient for practical purposes.
Since CPLEX^{TM}, like other packages, just rounds the value of the objective function for the optimal solution without any warning, it is easy to forget that the result might be inaccurate. Actually, the authors made precisely this mistake in a former version of this paper [29], and discovered the error only after CPLEX^{TM} reported an obviously false answer when testing a model of the towers of Hanoi [25].
There exist packages, based on constraint logic programming over rationals [24, 5], that solve mixed programming problems exactly. Unfortunately, some experiments show that the performance is not acceptable for medium-sized problems.
In the next section we present another algorithm to perform the trap test, which delivers good results in practical problems.

6. An iterative algorithm to perform the trap test

Given a trap Θ marked at the initial marking, we know that every reachable marking M has to mark Θ. In other words, every reachable marking M must satisfy the linear constraint

$$l(\Theta) \equiv \sum_{p \in \Theta} M(p) \geq 1$$

We can then perform the trap test in the following way:

- compute all traps of the net marked under the initial marking; for each of them, add its corresponding linear constraint to the generalised state equation;
- check if the resulting system of (in)equations is feasible; the Petri net passes the test if the system is infeasible.

Unfortunately, this procedure is inefficient, because it requires the explicit enumeration of all traps, whose number can be exponential in the size of the net. A first thought is that we only need to compute the minimal traps with respect to set inclusion: if we have $\Theta_1 \subset \Theta_2$ for two traps Θ_1 and Θ_2, then $l(\Theta_1)$ implies $l(\Theta_2)$, and so $l(\Theta_2)$ can be removed. However, it is easy to construct a family of nets for which not only the number of traps but also the number of minimal traps grows exponentially in the size of the net.
The solution to avoid the explicit enumeration is to compute traps only on demand. We maintain a set of linear constraints L, which initially contains the constraints of the generalised state equation, and follow an iterative procedure. At each step, we check if L is feasible. If not, then the procedure terminates with the result "the system passes the test". If there is a solution with marking M, then we check if there is an initially marked trap Θ such that M does not satisfy the constraint $l(\Theta)$. If there are no such traps, then the procedure terminates with the result "the system fails the test". If such a trap Θ exists, then we add $l(\Theta)$ to L, and iterate.
Consider for instance the Petri net of Figure 4. We perform the trap test for the property "no reachable marking marks p_1 and p_6 simultaneously", which is clearly co-linear. The generalised state equation has a solution with marking $M = \{p_1, p_6\}$.

We find the trap $\{p_2, p_3, p_4, p_5\}$, which is initially marked but unmarked at M. We add the constraint $M(p_2) + M(p_3) + M(p_4) + M(p_5) \geq 1$ to the generalised state equation, and find that the new set of constraints is infeasible. So the Petri net passes the test, and the property holds. Notice that even though the Petri net has four initially marked minimal traps, we only need one of them.

We do not know yet how to compute the traps required by the procedure. Given a marking M, define an *M-trap* as a trap that

- is marked at the initial marking M_0, and
- is unmarked at the marking M.

We need an algorithm that gets the Petri net (N, M_0) and M as input, and returns an M-trap, if it exists, or the message "no M-traps" otherwise. Observe that there may be more than one M-trap. The linear constraints corresponding to different M-traps may cut away more or less solutions of L not corresponding to reachable markings. The *quality* of an M-trap can then be defined as the "strength" of its corresponding linear constraint. M-traps of good quality lead to a small number of iterations in the procedure. We look for an algorithm with the best trade-off between quality and computational cost.

We investigate the following three strategies:

(1) *Maximal M-trap.* It follows immediately from the definition of trap and M-trap that the union of M-traps is again an M-trap. So there exists a unique maximal M-trap, and this is the M-trap computed by this strategy. Since any other M-trap is included in the maximal one, the maximal trap induces the weakest linear constraint, and so this strategy has the poorest quality. However, as shown below, it can be computed very efficiently.

(2) *Minimal M-trap.* As we have seen above, minimal traps with respect to set inclusion have better quality than non-minimal ones. This strategy computes a (nondeterministically selected) minimal trap.

(3) *Smallest M-trap.* This strategy computes an M-trap with a minimal number of places. Notice first that this trap must also be minimal under set inclusion. Smallest traps are not guaranteed to have better quality than any other minimal traps. However, they lead to linear constraints involving a minimal number of variables, and so this strategy is a reasonable heuristic.

The Petri net of Figure 6 illustrates the difference between these strategies. Let $M = \{p_5\}$. The maximal M-trap is $\{p_1, p_2, p_3, p_4\}$. Both $\{p_1, p_3, p_4\}$ and $\{p_1, p_2\}$ are minimal M-traps, but only the latter is a smallest M-trap.

We first examine the cost of the strategies, and then compare their performance on examples.

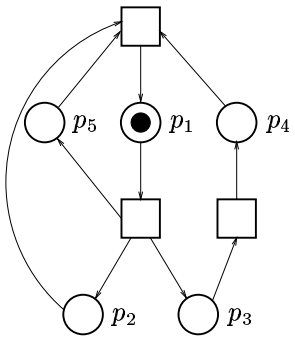

Figure 6. Maximal, minimal, and smallest M-traps

6.1. Maximal M-trap

The problem of computing a maximal M-trap can be easily reduced to the problem of computing the greatest fixpoint of a boolean function. For that, let $\{p_1, \ldots, p_n\}$ be the set of places of the net. We define the function $f: \{0,1\}^n \to \{0,1\}^n$ as follows:

$$f_i(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } M(p_i) = 1 \\ 0 & \text{if there is } (p_i, t) \in F \text{ such that } x_j = 0 \\ & \text{for every } p_j \in t^\bullet \\ 1 & \text{otherwise.} \end{cases}$$

where f_i is the i-th component of f. Intuitively, f can be seen as a function which takes (the characteristic vector of) a set of places and returns (the characteristic vector of) another set. A place is removed from the input set if it is marked at M or if there is a transition such that none of its output places belongs to the set. It is immediate to see that Q is a trap unmarked at M if and only if its characteristic vector χ_Q is a fixpoint of f, i.e. $f(\chi_Q) = \chi_Q$.

Since the mapping f is monotonic, it has a unique greatest fixpoint, which is the unique *maximal* trap Θ unmarked at M. If Θ is not marked at M_0, then, by the maximality of Θ, no M-traps exist. If Θ is marked at M_0, then it is an M-trap, and moreover the maximal M-trap. So computing the maximal M-trap reduces to computing the greatest fixpoint of f. A linear time algorithm for the greatest fixpoint is given by Andersen in [2][6] Since the set of clauses contains $O(|P| \cdot |T|)$ elements, and each clause contains at most $|P|$ literals, the maximal M-trap can be computed in $O(|P|^2 \cdot |T|)$ time.

As mentioned in [2], the linear time algorithm for the computation of the greatest fixpoint is similar to the "pebbling" algorithm of [16] and to the LTUR algorithm of [32] for deciding satisfiability of Horn formulae. Actually, it is observed in [33] that the computation of the maximal M-trap can also be reduced to this problem.

6.2. Minimal M-trap

Let $MaxTrap(N, Q)$ be an algorithm that returns the maximal trap of N containing no place of Q, and runs in $O(|P|^2 \cdot |T|)$ time. Let P_{M_0}, P_M be set of places marked at M_0 and M, respectively. The following algorithm computes a minimal M-trap with respect to set inclusion: (if there is no M-trap, then the algorithm returns the special value $\perp$):

```
1.   Θ := MaxTrap(N, P_M);
2.   if Θ ∩ P_{M_0} = ∅ then return ⊥ fi;
3.   Q := Θ;
4.   while Q ∩ Θ ≠ ∅ do
5.       choose p ∈ Q ∩ Θ;
6.       Θ' := MaxTrap(N, P_M ∪ {p});
7.       if Θ' ∩ P_{M_0} ≠ ∅ then Θ := Θ' fi;
8.       Q := Q \ {p}
9.   od;
10.  return Θ
```

Here, the variable Θ stores the current M-trap; Q stores the set of places p for which $\Theta \setminus \{p\}$ might still contain an M-trap.
Since $MaxTrap(N, Q)$ runs in $O(|P|^2 \cdot |T|)$ time and the **while**-loop is executed at most $|\Theta|$ times, the algorithm runs in $O(|P|^3 \cdot |T|)$ time.
In order to see how the choice of $p \in Q \cap \Theta$ has an effect on the size of the resulting trap, consider the following two computations of minimal M-traps of Figure 6. If we choose $p := p_2$, then the result is $\Theta = MaxTrap(N', \{p_5, p_2\}) = \{p_1, p_3, p_4\}$. If we choose p_3 then we set $\Theta := MaxTrap(N', \{p_5, p_3\}) = \{p_1, p_2, p_4\}$; the only useful choice is p_4, and we obtain the final result $\Theta = MaxTrap(N', \{p_5, p_3, p_4\}) = \{p_1, p_2\}$.

6.3. Smallest M-trap

The existence of polynomial time algorithms for this strategy is unlikely, because of the following result

THEOREM 3 *The following problem is NP-complete:*
Given: a Petri net (N, M_0), a marking M, a positive integer k,
To decide: does (N, M_0) contain an M-trap with at most k places?

Proof: Membership in NP is obvious. Hardness is shown by reduction from HITTING SET [19]. An instance of HITTING SET consists of a collection C of subsets of a finite set S, and a positive integer $K \leq |S|$. The problem is to decide whether there is a subset $S' \subseteq S$ with $|S'| \leq K$ such that S' contains at least one element from each subset in C.

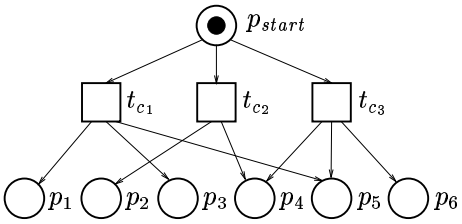

Figure 7. N_{HIT} for $S = \{1, \ldots, 6\}$ and $C = \{\{1, 3, 5\}, \{2, 4\}, \{4, 5, 6\}\}$.

Given an instance C, S, K of HITTING SET we construct in linear time an instance (N, M_0), M, k of our problem. Figure 7 shows (N, M_0) for the particular case $S = \{1, \ldots 6\}$ and $C = \{\{1, 3, 5\}, \{2, 4\}, \{4, 5, 6\}\}$. In general, N contains:

- a place p_s for each $s \in S$, plus a place p_{start},
- a transition t_c for each $c \in C$, with ${}^\bullet t_c = \{p_{start}\}$ and $t_c{}^\bullet = c$.

M_0 puts one token on p_{start} and no tokens anywhere else. M puts no tokens anywhere. Finally, $k = K$. It is immediate to see that S' contains one element from each subset in C if and only if the set $\{p_s \mid s \in S'\} \cup \{p_{start}\}$ is a trap of N. ■

The problem of computing a smallest M-trap can be reduced to an integer programming problem. Recall: If Q is an M-trap, then the assignment

$$Y(p) := \begin{cases} 1 & \text{if } p \in Q \\ 0 & \text{otherwise} \end{cases}$$

is a solution of the primal system. Conversely, if $\{Y(p) := a_p \mid p \in P\}$ is a solution, then the set $\{p \in P \mid a_p > 0\}$ is a M-trap. So we can compute a smallest M-trap as the integer solution of the primal system which minimises the objective function $\sum_{p \in P} Y(p)$.
In the next section we obtain experimental results that allow us to compare the performance of our three strategies (maximal, minimal and smallest M-traps).

6.4. Improving the iterative algorithm

The iterative procedure for the trap test can be improved. To explain why recall that branch-and-bound algorithms for solving (mixed) integer linear problems pick up a variable, branch adequately on its possible values, and examine each branch in turn. They construct in this way a tree whose nodes are sets of constraints; the set of constraints of a node is contained in the set of constraints of its of its children. The iterative procedure as presented at the beginning of this section *completely* solves a mixed programming problem after the addition of each new

linear constraint corresponding to each trap. A more flexible approach is to merge the computation of traps and the branch-and-bound algorithm. If the algorithm is currently checking the existence of solutions satisfying a certain set of constraints L, we can try to find a trap Θ such that $L \cup l(\Theta)$ is infeasible. Algorithms working in this way have been called *branch-and-cut*-algorithms [21]. In the next section we obtain some preliminary experimental results using this approach and the ABACUS system [41].

7. Experimental results on the trap test

In this section we show that the trap test can prove properties of interesting systems that are out of reach of the state equation test. We also show that the trap test can be efficiently performed using the iterative procedure of the last section. We compare the performances of the mixed and integer trap tests, as well as the performance of the three strategies for the computation of traps. We give preliminary results on the performance of a branch-and-cut algorithm. Finally, we compare these results with those obtained by means of conventional reachability tools.
All results have been obtained on a SUN SPARC 20/702 with 88 MB of memory. All integer and mixed programming problems have been solved using CPLEX^{TM} (version 3.0) [14], with the exception of those solved by the branch-and-cut algorithm. This algorithm has been implemented in the ABACUS system [41], and uses the CPLEX^{TM} library to solve linear programming problems.

7.1. The examples

We provide experimental results on the following systems:
Mutex algorithms. Six well-known mutual exclusion algorithms by De Bruijn, Dekker, Dijkstra, Knuth, Lamport and Peterson. The pseudocode description of the algorithms has been taken from [37]. The algorithms have been encoded in B(PN)^2 (Basic Petri net Programming Notation), an imperative concurrent programming language designed to have a simple Petri net semantics [7]. The programs have been automatically compiled into 1-safe Petri nets by the PEP-tool [6]. The compilation generates a number of redundant places and transitions. They have not been removed, because it is not easy to do so for an unexperienced user of the tool.
The algorithms by Dijkstra and Knuth have been scaled from 2 to 6 processes in order to obtain some data about the performance of the iterative procedure as a function of the size of the net.
A telephone protocol. A call handling for intelligent telephone networks, closely related to a *Basic Call State Model* [36] of the ITU–T (former CCITT) standardization committee. The system is described in [26]. We have used the B(PN)^2 translations of [20] for 2 and 3 telephones. Again, the PEP-tool has been used to generate a 1-safe Petri net.
A production cell. A production cell with two conveyor belts, a rotatable robot with two extendable arms, an elevatory rotary table, a metal press that forges

Table 1. Size of the examples

Example	Places	Transitions
De Bruijn	86	165
Dekker	50	75
Dijkstra 2	68	86
Dijkstra 3	104	156
Dijkstra 4	142	252
Dijkstra 5	182	380
Dijkstra 6	224	546
Knuth 2	78	137
Knuth 3	125	370
Knuth 4	178	945
Knuth 5	237	2426
Knuth 6	302	6553
Lamport	39	42
Peterson	40	69
Telefon 2	87	188
Telefon 3	232	672
Cell	231	202

plates, and a travelling crane. The system is described in [28], where it is used as benchmark for many different formal methods. The cell has been directly modelled as a 1-safe Petri net in [23], and so the net does not contain any redundant elements. The system is parameterized by the number of plates circulating; we consider the version with 5 plates.
Table 1 shows the sizes of the Petri nets for all systems.

7.2. The properties

We have proved deadlock-freedom of all the systems, mutual exclusion of the mutex algorithms, and the safety requirements 8.a.1, 8.a.2, 8.b.1, 8.b.2, 9, 10, and 15 of the production cell (for a full description of the requirements, see [28]), formalised as invariants. For instance, requirement 9, "a robot arm may only rotate if the arm is retracted", is formalised as the invariant

$$(\text{robot\_left} \vee \text{robot\_right}) \rightarrow (\text{arm1\_retract\_ext} \wedge \text{arm2\_retract\_ext})$$

where robot_left, robot_right, etc. hold when the place of the model with the same name holds a token. These invariants can be easily shown to be co-linear. For instance, the markings violating requirement 9 are those satisfying

$$\begin{aligned} M(\text{robot\_left}) + M(\text{robot\_right}) &\geq 1 \\ M(\text{arm1\_retract\_ext}) + M(\text{arm2\_retract\_ext}) &\leq 1 \end{aligned}$$

7.3. Tests, and algorithms used to perform them

We have applied the state equation and the trap tests in their mixed and integer versions. The trap test is performed by means of

- the iterative procedure described in the last section, using each of the three strategies for the computation of M-traps;
- ABACUS branch-and-cut algorithm, using the minimal M-trap strategy.

7.4. Qualitative results

Mutex algorithms. Deadlock-freedom can be proved for all algorithms but Dekker's using the state equation test. More precisely, all algorithms but Dekker's pass the mixed state equation test, but Dekker's algorithm fails both the mixed and integer tests. Dekker's algorithm passes the mixed trap test for deadlock-freedom. Mutual exclusion cannot be proved for any of the algorithms using the state equation test. This is not surprising, since the property depends on the ability of the processes to test the values of variables. Mutual exclusion can be proved for all algorithms using the mixed trap test.
Telephone protocol and production cell. None of the properties of the telephone protocol and the production cell can be proved with the state equation tests. All of them can be proved with the mixed trap test.

7.5. Quantitative results

Table 2 shows data on the results of the *mixed* trap test, performed with the iterative algorithm and the three different strategies: maximal, minimal, and smallest trap. The first colums give the property under consideration. The next three columns (each of them divided into two), present the number of iterations (i.e., the number of traps that had to be computed before the system of inequations became infeasible), and the total time in seconds for each of the strategies. Table 3 shows the same data for the *integer* test.
The main conclusions are:

- The strategies perform as expected with respect to the number of iterations: the maximal strategy is worse than the minimal, which is worse than the smallest. The minimal strategy is clearly the most efficient in average, although it is sometimes beaten by the smallest strategy. The maximal strategy cannot compete with the other two, neither in number of iterations - where it very often exceeds an (arbitrarily selected) upper bound - nor in time.
- The mixed test performs in most cases better than the integer test. However, the difference in performance is not very significant if the integer test is implemented carefully. Mixed programming problems are solved by branch-and-bound algorithms. Loosely speaking, these algorithms pick up a variable, branch adequately on its possible values, and examine each branch in turn. The integer test performs well if it branches on the variables of M first. If the algorithm branches on the variables of X first, then the performance is far worse.

Table 2. Performance of the mixed trap test.

Example	Property	Maximal		Minimal		Smallest	
		Iterations	Time	Iterations	Time	Iterations	Time
De Bruijn	Mutex	> 100	> 24	7	1	9	3
Dekker	Deadlock	11	4	2	1	5	2
	Mutex	23	2	5	< 1	5	1
Dijkstra 2	Mutex	> 100	> 19	15	1	7	2
Dijkstra 3	Mutex	> 100	> 31	31	4	16	7
Dijkstra 4	Mutex	> 100	> 40	51	11	23	12
Dijkstra 5	Mutex	> 100	> 40	71	25	35	30
Dijkstra 6	Mutex	> 100	> 54	124	212	58	216
Knuth 2	Mutex	> 100	> 24	8	1	9	3
Knuth 3	Mutex	> 200	> 150	14	3	19	13
Knuth 4	Mutex	> 200	> 269	24	8	31	43
Knuth 5	Mutex	> 200	> 553	34	29	48	157
Knuth 6	Mutex	> 200	> 919	42	204	64	640
Lamport	Mutex	12	1	5	1	5	1
Peterson	Mutex	25	2	7	1	6	1
Telefon 2	Deadlock	> 100	> 84	14	25	10	16
Telefon 3	Deadlock	> 500	> 4473	72	3384	32	742
Cell	Deadlock	> 100	> 91	24	75	26	125
	8.a.1	> 100	> 73	4	2	3	11
	8.a.2	> 100	> 63	5	2	5	17
	8.b.1	> 100	> 76	4	1	4	13
	8.b.2	> 100	> 79	5	2	8	19
	9	> 100	> 52	5	6	10	28
	10	> 100	> 65	11	3	14	46
	15	> 100	> 74	19	4	15	32

Table 3. Performance of the integer trap test.

Example	Property	Maximal		Minimal		Smallest	
		Iterations	Time	Iterations	Time	Iterations	Time
De Bruijn	Mutex	>100	> 27	10	2	12	4
Dekker	Deadlock	11	2	2	1	5	1
	Mutex	39	> 5	8	1	6	1
Dijkstra 2	Mutex	>100	> 20	14	2	10	3
Dijkstra 3	Mutex	>100	> 26	39	5	25	10
Dijkstra 4	Mutex	>100	> 32	57	13	34	22
Dijkstra 5	Mutex	>200	> 148	110	56	46	68
Dijkstra 6	Mutex	>300	> 420	146	204	69	321
Knuth 2	Mutex	>100	> 19	8	1	11	5
Knuth 3	Mutex	>200	> 463	17	3	19	13
Knuth 4	Mutex	>200	> 315	29	12	35	83
Knuth 5	Mutex	>300	> 1082	61	130	51	712
Knuth 6	Mutex	>500	> 5254	67	840	72	62288
Lamport	Mutex	19	1	5	1	5	1
Peterson	Mutex	34	4	9	1	6	1
Telefon 2	Deadlock	>300	> 925	14	24	10	25
Telefon 3	Deadlock	>500	> 7886	82	4146	66	1569
Cell	Deadlock	>200	> 288	25	125	25	98
	8.a.1	>100	> 62	8	3	11	27
	8.a.2	>100	> 56	9	2	8	18
	8.b.1	>100	> 71	8	2	11	36
	8.b.2	>100	> 64	9	3	10	19
	9	>100	> 54	8	5	9	22
	10	>100	> 70	10	3	11	39
	15	>100	> 74	15	3	14	32

Table 4. Performance of the branch-and-cut algorithm.

Example	Property	Nodes	Cuts	Time
De Bruijn	Mutex	1	7	< 1
Dekker	Deadlock	31	3	3
	Mutex	1	5	< 1
Peterson	Mutex	1	6	< 1
Dijkstra 2	Mutex	3	13	< 1
Dijkstra 3	Mutex	13	25	3
Dijkstra 4	Mutex	19	37	8
Dijkstra 5	Mutex	69	97	38
Dijkstra 6	Mutex	1431	1538	942
Knuth 2	Mutex	1	10	< 1
Knuth 3	Mutex	5	20	3
Knuth 4	Mutex	25	51	27
Knuth 5	Mutex	67	124	237
Knuth 6	Mutex	12923	13099	100778
Lamport	Mutex	1	6	< 1
Telefon 2	Deadlock	79	24	30
Telefon 3	Deadlock	599	188	1207
Cell	Deadlock	497	135	245
	8.a.1	1	4	1
	8.a.2	1	2	1
	8.b.1	1	5	1
	8.b.2	3	11	1
	9	1	3	1
	10	1	3	1
	15	1	4	1

Table 4 shows data on the results of ABACUS branch-and-cut algorithm with the minimal M-trap strategy, since it is the most efficient in average. For small examples this algorithm can very well compete with the results of the iterative version. However, in some of the larger examples (e.g., Dijkstra's mutex algorithm with more than four processes) the branching tree significantly blows up. The reason seems to be that the branching heuristics of ABACUS are not as good as those of CPLEX. The columns Nodes and Cuts contain the number of nodes of the generated tree, and the number of cuts used to prune it. The third column presents the time in seconds.

7.6. Comparison with tools for reachability analysis

We compare the performance of our technique with the performance of conventional tools for reachability analysis. We select the production cell example, and present results for:

- an algorithm that explicitely constructs the state space, implemented in PROD [43];

Table 5. Results for the production cell with PROD.

Property	Time[1]
Deadlock	1
8.a (1+2)	128
8.b (1+2)	104
9	29
10	80
15	84

- an algorithm that constructs a BDD representation of the state space, implemented in Petrify [12, 35] using Long's BDD package;

- a deadlock-checker and an LTL model-checker that construct a reduced state space by means of the stubborn set technique, also implemented in PROD.

All experiments were performed on the SUN Sparc machine used to compute the results of Tables 2, 3, and 4.
The model of the production cell has 1.66×10^6 states. PROD's algorithm for the explicit construction of the state space was aborted after 2 hours CPU time when it had consumed more than 100 MB of memory. Petrify constructed a BDD for the state space with 1.78×10^5 nodes, but only after five and a half hours. PROD's stubborn set algorithms turned out to be much more efficient. We proved deadlock-freedom with the deadlock-checker, and the safety requirements with the LTL model checker [42] (the same experiments had already been performed in [23], but on a different machine). The results are shown in Table 5 (time in seconds). The trap test is less efficient than PROD's deadlock checker, but it clearly beats the LTL model checker.
We remark that Corbett has carried out an study [10] comparing the performance of the INCA system, based on the state equation, with reachability tools like SMV and SPIN.

8. Conclusions

In this paper we have used traps, a well-known concept of net theory [38, 15], to extend the range of systems that can be verified using linear constraints. We have developed a trap test (a necessary condition) for safety properties whose negations can be expressed by means of linear inequations.
Our work has been motivated by the limitations of the state equation test, an otherwise succesful verification technique developed and studied in several papers. Despite its success in certain application areas (in particular the verification of deadlock freedom of ADA programs), the state equation test has severe limitations, particularly when applied to systems of sequential components communicat-

ing through shared variables. The trap test has been introduced to remove or at least palliate these problems.
We have shown that the computational cost of the state equation and the trap tests coincide: both are NP-complete. We have then looked for efficient algorithms to perform the trap test. As in the case of the state equation test, the trap test turns out to be reducible to a mixed programming problem. Although this result leads to an elegant trap inequation, very similar to the state equation, it cannot be used to develop an efficient algorithm for practical problems. The reason is that the trap inequation suffers from numerical instability for large systems. To overcome this problem we have developed an iterative procedure to perform the trap test. We have analysed the computational complexity of different variants of this procedure.
Using the trap test we have been able to prove different properties of a number of systems that could not be proved with the state equation test. In particular, we have shown that the trap test is powerful enough to prove mutual exclusion of the most popular mutual exclusion algorithms. We have also proved properties of less academic examples, namely a telephone protocol and a production cell. The Petri net models of these system have some hundreds of places and transitions. In the case of the production cell we compared our technique with other tools. The trap test proved to be much more efficient than algorithms based on the explicit construction of the state space or in its symbolic representation by BDDs. For most properties (although not for deadlock-freedom), the trap test also performed significantly better than the stubborn set technique implemented in the PROD tool.
We think that tests based on linear constraints have a place among verification techniques. Since they may fail to prove properties of very small systems, they work at best in combination with exact algorithms, like model-checkers. This guarantees good results for small systems, while the tests may prove properties for systems out of reach of the model-checker. Such a combination is currently running on a prototype of the PEP-tool [6].

Related work

The state equation has been applied, although not under this name, by Avrunin, Corbett and other authors [3, 4, 10, 11]. Since this work has been discussed at length in the paper, we do not consider it here any further.
Linear algebraic techniques have a long tradition in Petri net theory [31]. In particular, the state equation was introduced by Murata, and has been applied to different problems [8, 11, 34]. It is strongly related to place invariants, another well-known technique [38, 15, 18].
Techniques for the computation of traps based on linear algebra and logic are presented not only in [1] (the basis of Subsection 5.1), but also in [17, 27, 33]. These papers are however less suitable for the development of our trap test.
The idea of constructing a linear upper approximation of the set of reachable states was also proposed very early by Cousot and Halbwachs in the field of abstract interpretation [13], and was later studied further by Halbwachs [22]. However, this approach differs from ours in that we derive the linear approximation directly

from the structure of the system. On the contrary, in the abstract interpretation technique the linear approximation is constructed by starting with the initial state as seed and, loosely speaking, computing iteratively a linear upper approximation of the set of successors of the current set of states, until the approximation converges. The procedure is guaranteed to produce the best possible linear approximation of the set of states (its convex hull), but suffers more than our approach from the state explosion problem.

Acknowledgments

We thank Theodor Lettmann for suggesting to use HITTING SET, Stefan Thienel for his help with ABACUS, and Peter Deussen and Monika Heiner for sending us their source files of the production cell. Thanks go also to an anonymous referee for comments that helped to improve the presentation of the paper.

Notes

1. Notice the difference with the usual concept of testing a program: If the program passes the test, then it may be correct or incorrect; if it fails the test, then it is incorrect.
2. These papers do not use the name "state equation".
3. Since M is in fact a linear function of X, it would still be more general to define $\mathcal{P}$ as a linear predicate on the vectors X, i.e., $\mathcal{P}(X) \Leftrightarrow A \cdot X \geq b$, and this is in fact the approach of [11]. For our case studies the formulation of the text suffices.
4. For applications of this result see [30].
5. The branch-and-bound algorithm used to solve the integer problem should branch in the variables of M *before* branching on the variables of X.
6. Actually, the "chasing ones" algorithm of [2] computes the least fixpoint of f but the algorithm for the greatest fixpoint is completely dual.
7. The verification process of PROD runs in two phases (first compilation, and second stubborn set reduced reachability graph construction). The time given in the table refers to the second phase, while the compilation needs about four minutes.

References

1. H. Alaiwan and J.F. Toudic. Recherche des Semi-flots, des Verrous et des Trappes dans les Réseaux de Petri. *Technique et Science Informatique*, 4(1), pages 103–112, 1985.
2. H.R. Andersen. Model checking and boolean graphs *Theoretical Computer Science* 126, pages 3–30, 1994.
3. G.S. Avrunin, J.C. Corbett, and U.A. Buy. Integer Programming in the Analysis of Concurrent Systems. In K.G. Larsen and A. Skou, editors, *CAV '91*, volume 575 of *Lecture Notes in Computer Science*, pages 92–102, 1991.
4. G.S. Avrunin, U.A. Buy, J.C. Corbett, L.K. Dillon, and J.C. Wileden. Automated Analysis of Concurrent Systems with the Constrained Expression Toolset. *IEEE Transactions in Software Engineering*, 17(11), pages 1204–1222, 1991.
5. P. Barth and A. Bockmayr. Modelling Mixed-Integer Optimisation Problems in Constraint Logic Programming. Technical Report MPI-I-95-2-011, Max-Planck-Institut für Informatik, Saarbrücken, 1995.
6. E. Best and H. Fleischhack, editors. *PEP: Programming Environment based on Petri Nets*. Hildesheimer Informatikbericht 14/95, University of Hildesheim, Germany, 1995.

7. E. Best and R.P. Hopkins. $B(PN)^2$ – A Basic Petri Net Programming Notation. In *Proceedings of PARLE '93*, volume 694 of *Lecture Notes in Computer Science*, pages 379–390. Springer-Verlag, 1993
   Also: Hildesheimer Informatikbericht 27/92, University of Hildesheim, Germany, 1992.
8. G.V. Brams. *Réseaux de Petri: Theorie et Practique, Vols. I and II.* Masson, 1982.
9. A. Cheng, J. Esparza, and J. Palsberg. Complexity Results for 1-safe Petri Nets. *Theoretical Computer Science* 147, pages 117–136, 1995.
10. J.C. Corbett. Evaluating Deadlock Detection Methods for Concurrent Software. In T. Ostrand, editor, *Proceedings of the 1994 International Symposium on Software Testing and Analysis*, pages 204–215, New York, 1994.
11. J.C. Corbett and G.S. Avrunin. Using Integer Programming to Verify general Safety and Liveness properties. *Formal Methods in System Design*, 6(1), pages 97–123, 1995.
12. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3), pages 315–325, 1997. Technical report version available at http://www.lsi.upc.es/ jordic/petrify/refs/.
13. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints among Variables of a Program. In *5th ACM Symposium on Principles of Programming Languages*. ACM-Press, 1978.
14. CPLEX Optimization Inc. *Using the CPLEX^{TM} Callable Library and CPLEX^{TM} Mixed Integer Library.*
15. J. Desel and J. Esparza. *Free-choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.
16. W.F. Dowling and J.H. Gallier. Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *Journal of Logic Programming* 1, pages 267–284, 1984.
17. J. Ezpeleta, J.M. Couvreur, and M. Silva. A New Technique for Finding a Generating Family of Siphons, Traps and ST-Components. Application to Colored Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 674 of *Lecture Notes in Computer Science*, pages 126–147. Springer Verlag, 1993.
18. J. Desel. *Petrinetze, lineare Algebra und lineare Programmierung.* Teubner-Texte zur Informatik 26, 1998.
19. M.R. Garey and D.S. Johnson. *Computers and Intractability.* Freeman and Company, 1979.
20. B. Grahlmann. Verifying Telecommunication Protocols with PEP. In *Proceedings of RELECTRONIC '95, 9th Symposium on Quality and Reliability in Electronics.* Scientific Society for Telecommunications, pages 251–256, 1995.
21. M. Grötschel, M. Jünger and G. Reinelt. A Cutting Plane Algorithm for the Linear Ordering Problem. *Operations Research* 32, pages 1195–1220, 1984.
22. N. Halbwachs. About Synchronous Programming and Abstract Interpretation. In B. Le Charlier, editor, *SAS '94: Static Analysis Symposium*, volume 864 of *Lecture Notes in Computer Science*, pages 179–192. Springer-Verlag, 1994.
23. M. Heiner and P. Deussen. Petri Net Based Qualitative Analysis – a Case Study. Technical Report BTU Cottbus, I-08/1995, 1996. A short version appeared in: Petri Net Based Design and Analysis of Reactive Systems, in the Proc. of *WODES'96, Workshop on Discrete Event Systems*, Edinburgh, 1996.
24. C. Holzbaur. A Specialized, Incremental Solved Form Algorithm for Systems of Linear Inequalities. Technical Report Austrian Research Institute for Artificial Intelligence, Vienna, TR-94-07, 1994.
25. L. Jenner. Ein Prozedurkonzept für die parallele Hochsprache B(PN)^2. Master Thesis. University of Hildesheim, 1994.
26. S. Kleuker. A gentle Introduction to Specification Engineering using a Case Study in Telecommunications. In P.D. Mosses, M. Nielsen, and M.I. Schwartzbach, editors, *TAPSOFT '95*, volume 915 of *Lecture Notes in Computer Science*. Springer-Verlag, pages 636–650, 1995.
27. K. Lautenbach. Linear Algebraic Calculation of Deadlocks and Traps. In H.J. Genrich, K. Voss, and G. Rozenberg, editors, *Concurrency and Nets.* Springer-Verlag, pages 315–336, 1987.
28. C. Lewerentz and T. Lindner. Formal Development of Reactive Systems- Case Study Production Cell *Lecture Notes in Computer Science* 254. Springer-Verlag, 1995.

29. S. Melzer and J. Esparza. Checking System Properties via Integer Programming. In H.R. Nielson, editor, *ESOP '96*, volume 1058 of *Lecture Notes in Computer Science*, pages 250–264. Springer-Verlag, 1996.
30. S. Melzer and S. Römer. Deadlock Checking using Net Unfoldings. In O. Grumberg, editor, *CAV '97*, volume 1254 of *Lecture Notes in Computer Science*, pages 352–363. Springer-Verlag, 1997.
31. G. Memmi and G. Roucairol. Linear Algebra in Net Theory. In W. Brauer, editor, *Net Theory and Applications*, volume 84 of *Lecture Notes in Computer Science*, pages 213–223. Springer-Verlag, 1980.
32. M. Minoux. LTUR: a simplified linear-time unit resolution algorithm for Horn formulae and computer implementation. *Information Processing Letters* 29, pages 1–12, 1988.
33. M. Minoux and K. Barkaoui. Deadlocks and Traps in Petri nets as Horn-satisfiability Solutions and Some Related Polynomially Solvable Problems. *Discrete Applied Mathematics* 29, pages 195–210, 1990.
34. T. Murata. Petri nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4), pages 541–580, 1989.
35. E. Pastor, O. Roig, J. Cortadella, and R.M. Badia. Petri Net Analysis Using Boolean Manipulation. In Robert Valette, editor, *Application and Theory of Petri Nets 1994*, volume 815 of *Lecture Notes in Computer Science*, pages 416–435. Springer-Verlag, 1994.
36. CCITT Recommendations Q.1200. Intelligent Networks, final version. Technical report, 1992.
37. M. Raynal. *Algorithms for Mutual Exclusion*. North Oxford Academic, 1986.
38. W. Reisig. *Petri Nets*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, 1985.
39. A. Schrijver. *Theory of Linear and Integer Programming*. Series in Discrete Mathematics. Wiley, 1986.
40. P. Starke. *Analyse von Petri-Netz-Modellen*. Teubner, 1990.
41. S. Thienel. ABACUS–*A Branch And CUt System*. PhD thesis, University of Cologne, 1995.
42. A. Valmari. A Stubborn Attack on State Explosion. *Formal Methods in System Design* 1, pages 297–322, 1992.
43. K. Varpaaniemi, J. Halme, K. Hiekkanen, and T. Pyssysalo. *PROD reference manual*. Technical Report 13, B Series, Department of Computer Science, Helsinki University of Technology, 1995.