Proving Safety Properties of Infinite State Systems by Compilation into Presburger Arithmetic

Laurent Fribourg * Hans Olsén

LSV, E.N.S. Cachan & CNRS
61 av. Wilson, 94235 Cachan-France
fribourg@lsv.ens-cachan.fr

IDA, Linköping University
S-58183 Linköping-Sweden
hanol@ida.liu.se

Abstract. We present in this paper a method combining path decomposition and bottom-up computation features for characterizing the reachability sets of Petri nets within Presburger arithmetic. An application of our method is the automatic verification of safety properties of Petri nets with infinite reachability sets. Our implementation is made of a decomposition module and an arithmetic module, the latter being built upon Boudet-Comon's algorithm for solving the decision problem for Presburger arithmetic. Our approach will be illustrated on three nontrivial examples of Petri nets with unbounded places and parametric initial markings.

1 Introduction

We are interested in this paper in proving safety properties of infinite state systems. We will focus on Petri nets although our approach is applicable to other discrete models of concurrent systems such as automata with counters (see, e.g., [10]). There will be two sources of infinity for the state space of Petri nets that we will consider: the first one is the unboundedness of some places of the net; the second one comes from the fact that the initial marking of the net may contain parameters, thus representing an infinite family of markings. The safety properties that we will consider, will be merely of the form $\overline{x} \in lfp \Rightarrow I(\overline{x})$, where $\overline{x}$ represents a marking, lfp represents the set of reachable markings of the Petri net, and $I(\overline{x})$ a simple arithmetic relation characteristic of the safety property to be proved. Our method consists in characterizing the reachability relation $\overline{x} \in lfp$ as a formula $\xi(\overline{x})$ belonging to Presburger arithmetic (i.e., arithmetic without $\times$), then to prove $\xi(\overline{x}) \Rightarrow I(\overline{x})$ using a decision procedure for Presburger arithmetic [16]. The objective of our work is similar to the one of Hiraishi [11]. However Hiraishi constructs the arithmetic characterization of the set of reachable markings in a bottom-up manner by refining Karp-Miller's method for constructing coverability trees [14]. In contrast, our arithmetic characterization $\xi(\overline{x})$ is constructed using basically a structural method of path decomposition

* Part of this work was done while the author was visiting IASI-CNR (Roma), supported by HCM-Network CHRX.CT.930414 and IASI-CNR.

[18, 8]. Nevertheless, as explained hereafter in the paper, we will also integrate into the procedure some forward (bottom-up) computation routines that will speed up the arithmetic construction by propagating the initial marking values and testing the invariance of the intermediate constructed formulas.

2 Preliminaries

It is convenient to see the reachability problem for Petri nets as a least fixed-point problem for a certain class of logic programs with constraints over the integers domain $\mathcal{Z}$ [12]. These programs are of the form:

$$\begin{array}{lll} & p(\overline{x}) & \leftarrow\ B(\overline{x}). \\ r_1: & p(\overline{x}+\overline{t}_{r_1}) & \leftarrow\ \overline{x} > \overline{a}_{r_1},\ p(\overline{x}). \\ & \vdots & \\ r_m: & p(\overline{x}+\overline{t}_{r_m}) & \leftarrow\ \overline{x} > \overline{a}_{r_m},\ p(\overline{x}). \end{array}$$

where $\overline{x}$ is a vector of variables ranging over $\mathcal{Z}^n$, for some n, $B(\overline{x})$ a linear integer relation (relation defined by a Presburger formula), $\overline{t}_{r_i} \in \mathcal{Z}^n$ is a vector of constants, and $\overline{a}_{r_i}$ is a vector of constants belonging to $(\mathcal{Z} \cup \{-\infty\})^n$. As usual, $z > -\infty$, $z \neq -\infty$ and $-\infty \pm z = z \pm (-\infty) = -\infty$ for any integer $z \in \mathcal{Z}$, and $-\infty \geq -\infty$. For any vectors $\overline{x}_1$ and $\overline{x}_2$, we define $\overline{x}_1 > \overline{x}_2$ (resp. $\overline{x}_1 \geq \overline{x}_2$) to hold, if and only if the inequalities hold componentwise. The expression $max(\overline{x}_1, \overline{x}_2)$ denotes the vector obtained by taking the maximum of $\overline{x}_1$ and $\overline{x}_2$ componentwise. Since $z > -\infty$ holds for any $z \in \mathcal{Z}$, any constraint of the form $x > -\infty$, is simply considered as *true*.
One can see these programs as classical programs with counters expressed under a logic programming form. These programs have thus the power of expressivity of Turing machines. Henceforth we will refer to this class of programs as *programs with $\mathcal{Z}$-counters*. In the next section, we will see how these programs naturally encode the *reachability problem* for Petri nets (with inhibitors).
We now introduce a convenient description of the forward (or bottom-up) execution of programs with $\mathcal{Z}$-counters. A clause r of the form: $p(\overline{x}+\overline{t}_r) \leftarrow \overline{x} > \overline{a}_r, p(\overline{x})$ will be characterized by a couple $\langle \overline{t}_r, \overline{a}_r \rangle$. We say that $\overline{x}'$ is reachable from $\overline{x}$ via a clause $r : \langle \overline{t}_r, \overline{a}_r \rangle$, and denote $\overline{x} \xrightarrow{r} \overline{x}'$, if: $\overline{x}' = \overline{x} + \overline{t}_r \ \wedge\ \overline{x} > \overline{a}_r$. More generally, let $\Sigma = \{r_1, \ldots, r_m\}$. A sequence $w \in \Sigma^*$ is called a *path*. A path w is characterized by a couple $\langle \overline{t}_w, \overline{a}_w \rangle$ where $\overline{t}_w$ and $\overline{a}_w$ are recursively defined by respectively:

$$\begin{aligned} \overline{t}_\varepsilon &= \overline{0} \\ \overline{t}_{rw} &= \overline{t}_r + \overline{t}_w \end{aligned}$$

$$\begin{aligned} \overline{a}_\varepsilon &= -\overline{\infty} \\ \overline{a}_{rw} &= max(\overline{a}_r, \overline{a}_w - \overline{t}_r) \end{aligned}$$

We say that $\overline{x}'$ is reachable from $\overline{x}$ via a path $w : \langle \overline{t}_w, \overline{a}_w \rangle$, and denote $\overline{x} \xrightarrow{w} \overline{x}'$, if: $\overline{x}' = \overline{x} + \overline{t}_w \ \wedge\ \overline{x} > \overline{a}_w$. Given two paths w_1 and w_2, it follows from the above definition that: $\overline{x} \xrightarrow{w_1 w_2} \overline{x}'$ iff $\exists \overline{x}'' :\ \overline{x} \xrightarrow{w_1} \overline{x}'' \wedge \overline{x}'' \xrightarrow{w_2} \overline{x}'$. Given a language $L \subseteq \Sigma^*$, we say that $\overline{x}'$ is reachable from $\overline{x}$ via L, and denote $\overline{x} \xrightarrow{L} \overline{x}'$, if $\exists w \in L: \ \overline{x} \xrightarrow{w} \overline{x}'$. As usual, the reflexive-transitive closure of relation $\xrightarrow{L}$ is denoted $\xrightarrow{L^*}$. We also write $\overline{x} \xrightarrow{L_1} \overline{x}'' \xrightarrow{L_2} \overline{x}'$, instead of $\overline{x} \xrightarrow{L_1} \overline{x}'' \ \wedge \ \overline{x}'' \xrightarrow{L_2} \overline{x}'$. From the definitions above, we immediately get:

Proposition 1. *For any path $w \in \Sigma^*$ and any languages $L_1, L_2 \subseteq \Sigma^*$. We have:*

1. $\overline{x} \xrightarrow{L_1 L_2} \overline{x}' \ \Leftrightarrow \ \exists\ \overline{x}'' : \ \overline{x} \xrightarrow{L_1} \overline{x}'' \xrightarrow{L_2} \overline{x}'$
2. $\overline{x} \xrightarrow{w^*} \overline{x}' \ \Leftrightarrow \ \exists\ k \geq 0: \ \overline{x}' = \overline{x} + k \cdot \overline{t}_w \ \wedge \ \forall\ 0 \leq k' < k: \ \overline{x} + k' \cdot \overline{t}_w > \overline{a}_w$

Note, in the last equivalence, that if $k = 0$, then $\overline{x} = \overline{x}'$ and $\forall\ 0 \leq k' < k: \overline{x} + k' \cdot \overline{t}_w > \overline{a}_w$ is vacuously true. It is easy to see that, for $k > 0$, the universally quantified subexpression is equivalent to $\overline{x} + (k-1) \cdot \overline{t}_w^- > \overline{a}_w$ where $\overline{t}_w^-$ is the vector obtained from $\overline{t}_w$ by letting all nonnegative components be set to zero. Therefore, the whole equivalence becomes:

2'. $\overline{x} \xrightarrow{w^*} \overline{x}' \ \Leftrightarrow \ \overline{x}' = \overline{x} \ \vee \ \exists\ k > 0: \ \overline{x}' = \overline{x} + k \cdot \overline{t}_w \ \wedge \ \overline{x} + (k-1) \cdot \overline{t}_w^- > \overline{a}_w$

As a consequence, given a path w, the relation $\overline{x} \xrightarrow{w^*} \overline{x}'$ is actually an *existentially* quantified formula of Presburger arithmetic having $\overline{x}$ and $\overline{x}'$ as free variables. More generally, define a *flat* language as a language of the form $w_1^* .... w_c^*$ where each w_i $(1 \leq i \leq c)$ is a path [2]. By proposition 1 it follows that the relation $\overline{x} \xrightarrow{L} \overline{x}'$ for a flat language L, can be expressed as an existentially quantified formula of Presburger arithmetic, having $\overline{x}$ and $\overline{x}'$ as free variables. More precisely, the reachability relation $\overline{x} \xrightarrow{L} \overline{x}'$ is expressed as a disjunction of a number of matrix expressions of the form: $\exists\ \overline{k}_i : \ \overline{x}' = \overline{x} + C_i \overline{k}_i \ \wedge \ \overline{x} + D_i \overline{k}_i > \overline{e}_i$. where C_i and D_i are matrices, and $\overline{e}_i$ some vector of constants. Such a formula can be simplified as a quantifier-free formula, say $\zeta_L(\overline{x}, \overline{x}')$, by elimination of the existentially quantified variables $\overline{k}_i$ through a Presburger decision procedure (see [16]).
Given a program with $B(\overline{x})$ as a base case and recursive clauses Σ, the least fixed-point of its immediate consequence operator (see [12][13]), which is also the least $\mathcal{Z}$-model, may be expressed as: $lfp = \{\ \overline{x}' \mid \exists \overline{x}: \ B(\overline{x}) \ \wedge \ \overline{x} \xrightarrow{\Sigma^*} \overline{x}'\ \}$. Our aim is to characterize the membership relation $\overline{y} \in lfp$ as a quantifier-free formula having $\overline{y}$ as free variables. In order to achieve this, our approach here is to find a flat language $L \subseteq \Sigma^*$, such that the following equivalence holds: $\overline{x} \xrightarrow{\Sigma^*} \overline{x}' \ \Leftrightarrow \ \overline{x} \xrightarrow{L} \overline{x}'$. An arithmetic characterization of $\overline{y} \in lfp$ is then: $\exists \overline{x}\ B(\overline{x}) \wedge \zeta_L(\overline{x}, \overline{y})$. Such a formula can be in turn simplified as, say $\xi_{B,L}(\overline{y})$, by elimination of $\overline{x}$ again through a Presburger decision procedure.
Given a formula $\xi(\overline{x})$ and a path w, we call *w-closure* of $\xi(\overline{x})$, a quantifier-free formula $\xi'(\overline{x})$ obtained from $\exists \overline{z}: \quad \xi(\overline{z}) \wedge \overline{z} \xrightarrow{w^*} \overline{x}$ by elimination of all the existential variables (*viz.*, $\overline{z}$ and variable k implicit in $\overline{z} \xrightarrow{w^*} \overline{x}$). We say that *a path w lets invariant* a formula ξ if $\xi(\overline{x}) \wedge \overline{x} \xrightarrow{w} \overline{x}'$ implies $\xi(\overline{x}')$, for all $\overline{x}, \overline{x}'$. We say that *a set Σ lets invariant* ξ, and write 'invariant(ξ,Σ)', if every element

[2] A close (but slightly different) notion has been introduced by Ginsburg [9] under the name of 'bounded language'.

of Σ lets invariant ξ. In the following, given a formula ξ and a language L, we will often abbreviate an expression of the form $\xi(\overline{x}) \wedge \overline{x} \xrightarrow{L} \overline{x}'$ as $\xi(\overline{x}) \xrightarrow{L} \overline{x}'$. The w-closure of a formula ξ will be accordingly denoted as $\xi \xrightarrow{w^*}$. Note that ξ always implies $\xi \xrightarrow{w^*}$, and that the converse holds iff w lets invariant ξ.

3 Encoding of the reachability problem of Petri Nets

Consider a Petri net with n places and m transitions. In this section, we sketch out how to encode the reachability problem for Petri nets, via an n-ary predicate p defined by a program with $\mathcal{Z}$-counters. Each place j $(1 \leq j \leq n)$ of the Petri net will be encoded as an arithmetic variable x_j. A marking is encoded as a tuple $\langle b_1, ..., b_n \rangle$ of n nonnegative integers. (The value b_j represents the number of tokens contained in place j.) Each transition i $(1 \leq i \leq m)$ will be encoded as a recursive clause r_i. An atom of the form $p(b_1, ..., b_n)$ means that a marking $\langle b_1, ..., b_n \rangle$ is reachable from the initial marking. The predicate p is defined as follows:

- The base clause r_0 is of the form:
  $p(x_1, ..., x_n) \leftarrow x_1 = b_1^0, ..., x_n = b_n^0.$
  where $\langle b_1^0, ..., b_n^0 \rangle$ denotes the initial marking.
- The clause r_i $(1 \leq i \leq m)$, coding for the i-th transition, is of the form:
  $p(x_1 + t_{i,1}, ..., x_n + t_{i,n}) \leftarrow \phi_i(x_1, ..., x_n), p(x_1, ..., x_n).$
  Here $t_{i,j}$ is the sum of the weights of the output arrows from transition i to place j, minus the sum of the weights of the input arrows from place j to transition i. The expression $\phi_i(x_1, ..., x_n)$ is of the form: $x_{j_1} > a_{j_1} - 1 \wedge ... \wedge x_{j_{c_i}} > a_{j_{c_i}} - 1 \wedge x_{k_1} = 0 \wedge \ldots \wedge x_{k_{d_i}} = 0$, where $j_1, ..., j_{c_i}$ are the input places of transition i, a_{j_α} is the weight of the arc going from place j_α to transition i $(1 \leq \alpha \leq c_i)$, and $k_1, ..., k_{d_i}$ are the inhibitors places of transition i. (The condition ϕ_i expresses that the i-th transition is enabled.)

A priori such a program does not belong to the class we consider due to the constraints of the form $x_{k_\alpha} = 0$ $(1 \leq \alpha \leq d_i)$. However by adding extra arguments, say x'_{k_α} $(1 \leq \alpha \leq d_i)$, which are initialized with 1 minus the initial value of x_{k_α}, and are incremented (resp. decremented) when x_{k_α} is decremented (resp. incremented), one can replace the constraint $x_{k_\alpha} = 0$ with $x'_{k_\alpha} > 0$. [3]
The least fixed-point *lfp* associated with the program corresponds to the *reachability set* associated with the Petri net, i.e. the set of all the markings reachable from the initial marking.
Sometimes it is interesting to reason generically with some *parametric initial markings*, i.e., initial markings where certain places are assigned parameters instead of constant values. This defines a family of Petri nets, which are obtained by replacing successively the parameters with all the possible nonnegative values. One can easily encode the reachability relation for a Petri net with a parametric initial marking via a program with $\mathcal{Z}$-counter by adding the initial marking

[3] By construction, x'_{k_α} is always equal to $1 - x_{k_\alpha}$, and may thus take negative values.

parameters as extra arguments of the encoding predicate. In the case of a Petri net with an initial marking containing a tuple of parameters, say $\overline{q}$, our aim is to characterize the relation $\overline{y} \in lfp$ as an arithmetical formula $\xi(\overline{q}, \overline{y})$ having $\overline{q}$ and $\overline{y}$ as free variables. This will allow us to determine all the values of the parameters $\overline{q}$ for which a given safety property holds (see sections 7.1, 7.2).

Example 1. Consider the Petri net in figure 1. (This example is the "swimming-pool" net from M. Latteux, see [3, 6].) With the initial marking $x_1 = x_2 = x_3 =$

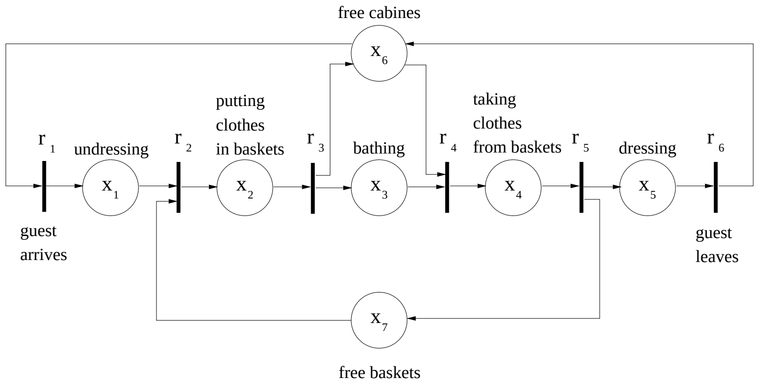

Figure 1

$x_4 = x_5 = 0$, $x_6 = q_1$ and $x_7 = q_2$ for some nonnegative parameters q_1 and q_2, the task is to show that there exists a deadlock regardless of what q_1 and q_2 are. The program encoding the reachability problem for this net is the following:

$r_0: \quad p(q_1, q_2, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \leftarrow$
$$x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 0, x_6 = q_1, x_7 = q_2.$$
$r_1: \quad p(q_1, q_2, x_1 + 1, x_2, x_3, x_4, x_5, x_6 - 1, x_7) \leftarrow x_6 > 0,$
$$p(q_1, q_2, x_1, x_2, x_3, x_4, x_5, x_6, x_7).$$
$r_2: \quad p(q_1, q_2, x_1 - 1, x_2 + 1, x_3, x_4, x_5, x_6, x_7 - 1) \leftarrow x_1 > 0, x_7 > 0,$
$$p(q_1, q_2, x_1, x_2, x_3, x_4, x_5, x_6, x_7).$$
$r_3: \quad p(q_1, q_2, x_1, x_2 - 1, x_3 + 1, x_4, x_5, x_6 + 1, x_7) \leftarrow x_2 > 0,$
$$p(q_1, q_2, x_1, x_2, x_3, x_4, x_5, x_6, x_7).$$
$r_4: \quad p(q_1, q_2, x_1, x_2, x_3 - 1, x_4 + 1, x_5, x_6 - 1, x_7) \leftarrow x_3 > 0, x_6 > 0,$
$$p(q_1, q_2, x_1, x_2, x_3, x_4, x_5, x_6, x_7).$$
$r_5: \quad p(q_1, q_2, x_1, x_2, x_3, x_4 - 1, x_5 + 1, x_6, x_7 + 1) \leftarrow x_4 > 0,$
$$p(q_1, q_2, x_1, x_2, x_3, x_4, x_5, x_6, x_7).$$
$r_6: \quad p(q_1, q_2, x_1, x_2, x_3, x_4, x_5 - 1, x_6 + 1, x_7) \leftarrow x_5 > 0,$
$$p(q_1, q_2, x_1, x_2, x_3, x_4, x_5, x_6, x_7).$$

4 Construction of Reachability Sets

Let us consider a program defined by a set of transitions $\Sigma_{original} : \{r_1, ..., r_m\}$. In order to characterize the relation $\overline{x} \xrightarrow{\{r_1, \ldots, r_m\}^*} \overline{x}'$, we will construct a sequence

$\{L_i\}_i$ of subsets of $\{r_1, ..., r_m\}^*$ which are "reachably-equivalent" to $\{r_1, ..., r_m\}^*$ in the sense that, for any $\overline{x}$ and $\overline{x}'$: $\overline{x} \xrightarrow{\{r_1,...,r_m\}^*} \overline{x}' \Leftrightarrow \overline{x} \xrightarrow{L_i} \overline{x}'$, and such that the last language in the sequence is flat. Such a flat language $L \subseteq \{r_1, ..., r_m\}^*$ will be generated by applying repeatedly a set of *decomposition* rules. Schematically, each decomposition rule, when applied to a set Σ, transforms it into a list Δ of sets of the form $[\Sigma_1, \Sigma_2, \ldots, \Sigma_c]$ such that Σ^* is reachably-equivalent to the language $\Sigma_1^* \Sigma_2^* \ldots \Sigma_c^*$. Every element of Σ_i $(1 \leq i \leq c)$ is either an element r of Σ, or is a path w obtained by composition of several elements of Σ. The process of decomposition is iterated on the list Δ: one set Σ_j of Δ is selected, and the list resulting from its decomposition is inserted in place of it within Δ, thus generating a new sequence Δ'. The process is iterated until either:

- all the sets $\Sigma_1, ..., \Sigma_c$ of the current list Δ are singletons of the form $\{w_1\}, \{w_2\}, \cdots, \{w_c\}$. This means that the language $\Sigma_1^* \Sigma_2^* ... \Sigma_c^*$ associated with Δ is flat (*termination with success*), or
- no decomposition rule applies onto the selected set Σ_j of list Δ (*termination with failure*).

Note that the process cannot loop forever because each decomposition rule transforms a set Σ into a sequence of sets of "lower dimension" [8]. The number of rules of decomposition is 5. They are: stratification, monotonic transition, monotonic guard, cyclic post-fusion and cyclic pre-fusion (see [8] for details). They are tried in this order, and the first that succeeds is applied. When a flat language $L:\ w_1^* \ldots w_c^*$ has been generated, a decision procedure for Presburger arithmetic is invoked in order to construct the formula $\xi_{B,L}$ (see section 2). Starting from the base case relation B, the arithmetic decision procedure computes $\xi_{B,L}$ by extending B with the successive w_i-closures $(1 \leq i \leq c)$. Formally, $\xi_{B,L}(\overline{x}')$ is defined to be $\xi_c(\overline{x}')$ where $\xi_0(\overline{x}')$ is $B(\overline{x}')$, and $\xi_{i+1}(\overline{x}')$ is a quantifier-free formula obtained from $\exists \overline{x} :\ \xi_i(\overline{x}) \ \wedge\ \overline{x} \xrightarrow{w_{i+1}^*} \overline{x}'$ by elimination of the existential variables $\overline{x}$ and k_{i+1} (implicit in $\overline{x} \xrightarrow{w_{i+1}^*} \overline{x}'$). Actually a more efficient system is implemented by invoking earlier the arithmetic decision procedure, and starting to construct $\xi_{B,L}$ during the decomposition process, without waiting for the flat language L to be fully generated. This is explained in the next section. Sometimes, it is interesting to keep track of the number of times k_i each w_i is repeated inside sequences of the form $w_1^* \ldots w_c^*$. In such cases one may construct a formula $\xi_c(\overline{x}', k_1, ..., k_c)$ where $\xi_0(\overline{x}')$ is $B(\overline{x}')$, and $\xi_{i+1}(\overline{x}', \overline{k}_i, k_{i+1})$ is a quantifier-free formula equivalent to $\exists \overline{x} :\ \xi_i(\overline{x}, \overline{k}_i) \ \wedge\ \overline{x} \xrightarrow{w_{i+1}^{k_{i+1}}} \overline{x}'$. This is useful when one wishes to exhibit some "counter-example" path $w_1^{k_1} \cdots w_c^{k_c}$ which ends at a marking that violates the safety property under study (see 7.1).

5 General Description of the System

Our system consists of a decomposition procedure and a decision procedure for Presburger arithmetic. We will represent the sequence Δ of decomposed languages as a list. Initially, Δ contains a single element: $\Sigma_{original}$. At each step, the *leftmost element* (head) of Δ is selected for further decomposition. The system builds up a formula ξ, which will eventually characterize the least fixed-point. This formula is initialized with the base case relation B, and is extended by Σ-closure whenever the head Σ of Δ is a singleton. Before attempting any decomposition onto Σ, one checks whether it lets invariant ξ (because there is no point in decomposing a set of transitions that will not yield anything new). The top loop of our procedure is thus as follows:

$\xi := B;\ \ \Delta := [\Sigma_{\text{original}}];$
while not empty(Δ) **do**
  $\Sigma :=$ head(Δ); $\Delta :=$ tail(Δ);
  if not invariant(ξ, Σ) **then**
    if singleton(Σ) **then** $\xi := \xi \xrightarrow{\Sigma^*}$
    else $\Delta :=$ decompose(Σ) $\otimes\ \Delta$
    fi
  fi
od

where Δ is a list of sets of transitions, ξ is a Presburger formula, and $\otimes$ is append. The arithmetic form $\xi_{B,L}$ of the least fixed-point is given by the exit value of ξ when executing the program. Henceforth, we will denote this exit formula by ξ_{final}. The associated flat language L is the composed sequence $\Sigma_1^* ... \Sigma_c^*$ where the Σ_is are the successive singletons used during the program execution for extending ξ by closure (step: $\xi := \xi \xrightarrow{\Sigma^*}$). The language L "covers" all the reachable markings of the net in the sense that: $B(\overline{x}) \wedge \overline{x} \xrightarrow{L} \overline{x}' \ \Leftrightarrow\ B(\overline{x}) \wedge \overline{x} \xrightarrow{\Sigma^*_{original}} \overline{x}'$. The invariance check before attempting decomposition is important since it allows to discard a lot of sets Σ of transitions, and shortens considerably the length of the computed flat language.

5.1 Invariance check

As mentioned, before attempting to decompose a set of transitions, we first check whether all the transitions in the head language Σ let the current arithmetic formula ξ computed so far, invariant. If this is the case, the set is simply dropped and attention is moved to the next set. That is, before decomposing Σ, we check whether $\xi(\overline{x}) \xrightarrow{\Sigma} \overline{x}' \ \Rightarrow\ \xi(\overline{x}')$ holds (see section 2). This is *a priori* a computationally expensive (space-exponential) test. However, by storing in a set $\Im$ those transitions that have been discovered to keep ξ invariant, a lot of redundant computations are avoided. Consider for example a list Δ made of the set $\{w_1, w_2, w_3\}$. Before trying to decompose $\{w_1, w_2, w_3\}$ we test the invariance of ξ for each of the transitions w_1, w_2, w_3. Assume that at least one of the three fails to let $\xi(\overline{x})$ invariant and that the decomposition rule of "monotonic transition" (see [8]) applies to w_2, say. At the next step we have to consider the list $\Delta' : [\{w_1, w_3\}, \{w_2\}, \{w_1, w_3\}]$, and have to test ξ for invariance through the head language $\{w_1, w_3\}$. But invariance of ξ through w_1 and w_3 has already been tested, so the invariance check consists at this point in a simple table look up.

When computing a w-closure of ξ, the information in $\Im$ is *a priori* lost, and the new formula ξ' has a new set $\Im'$ of invariant transitions, which should be constructed. Here again, a lot of costly invariance tests can be saved by observing that a transition, say v, of $\Im$, which commutes with w is guaranteed to be still in $\Im'$. This is formally justified by the following (easily provable):

Proposition 2. *Suppose:*

1. $\xi(\overline{x}) \xrightarrow{v} \overline{x}' \;\Rightarrow\; \xi(\overline{x}')$	invariance of v
2. $\xi'(\overline{x}') \;\equiv\; \exists \overline{x} : \; \xi(\overline{x}) \xrightarrow{w^*} \overline{x}'$	w-closure
3. $\overline{x} \xrightarrow{wv} \overline{x}' \;\Rightarrow\; \overline{x} \xrightarrow{vw} \overline{x}'$	commutation

Then invariance of v is preserved by ξ', i.e.: $\quad \xi'(\overline{x}) \xrightarrow{v} \overline{x}' \;\Rightarrow\; \xi'(\overline{x}')$.

By inspecting the definition of $\overline{x} \xrightarrow{wv} \overline{x}'$, and $\overline{x} \xrightarrow{vw} \overline{x}'$, one can see that the commutation check 3 of proposition 2 reduces to verifying a number inequalities among constants, which is computationally cheap. Transitions of $\Im$ that fail the commutation check usually turn out to have lost their invariance.

5.2 Failure of decomposition

So far in this section, we have assumed that the procedure of decomposition always succeeds. This may not be the case. In case of failure (i.e., when no rule of decomposition applies to the current set Σ), our strategy consists to remove some transitions from Σ according to some heuristics (essentially, random choice) until some decomposition rule applies or Σ becomes a singleton. This removal endangers the completeness of the finally generated formula ξ_{final} in the sense that it may not correspond any longer to a fixed-point. In such a case (i.e., when a transition of the original language $\Sigma_{original}$ does not let invariant ξ_{final}) the system detects it, and the whole procedure of fixed-point computation restarts with ξ_{final} taken as a new base case formula (in place of B). This process is iterated until a fixed-point is actually reached. (There is no guarantee that such a fixed-point will be reached as the process may now loop forever.) An example of such a process with transition removal and restarting, is given in section 7.3.

5.3 Space Explosion

Our underlying decomposition strategy allows to alleviate the problem of space explosion that immediately occurs with naive methods based on exhaustive state exploration (in the case of finite state systems). This is because, among all the paths that go from a generic marking to another one, only a reduced number of "representative" paths is retained when applying the decomposition strategy (see [18]). This path selectivity is reinforced through interaction with the arithmetic module because quantities of (invariant) transitions are discarded. Naturally, even if our method allows us to treat automatically some examples that are usually done by hand (see section 7), we also have to face quickly with a space explosion problem. From a theoretical point of view, this may be explained by the worst-case complexity of our procedure, which is space-superexponential due to the exponential space-complexity of the operation of w-closure (quantifier elimination) and the fact that the size of the language (number of w-closures) grows itself exponentially during the process of decomposition. This space explosion phenomenon is particularly sensitive when one deals with Petri nets having more than one parameter in their initial markings. A solution for overcoming the problem is sometimes to reduce the original Petri net into a simpler net through transformation rules, as those of Berthelot [1], which preserve basic safety properties (e.g., deadlock-freeness, boundedness). An example of such a preliminary net transformation is given in section 7.1.

6 Arithmetic Module

The decision procedure for Presburger arithmetic that we have implemented is Boudet-Comon's algorithm [2]. It has turned out to be very well suited for our needs. Given a system of equations and inequations, the Boudet-Comon algorithm generates a finite state automaton recognising the language of all solutions written as strings of binary digits. This algorithm has nearly optimal worst case complexity and behaves according to our experience very well in practice. One of the advantages is its simplicity. Variable elimination, conjunction, disjunction, negation and inclusion are all achieved by standard automata theoretic methods such as projection, intersection, union, complement and emptiness testing. Another advantage of Boudet-Comon method is that, due to its simplicity and generality, it is easy to construct specialized programs for computing specific relations on its top, or to store information during its execution. We have exploited this feature for making easier the proof of general safety properties such as boundedness and detection of deadlock, as explained hereafter.
Detecting unboundedness is achieved by investigating wether the reachability set is finite or infinite which is done efficiently by investigating the loops in Boudet-Comon's automaton. A deadlock in a Petri net may defined by:

$$\mathrm{deadlock}(\overline{q},\overline{x}) \equiv \xi_{final}(\overline{q},\overline{x}) \wedge \mathrm{no\_transition\_enabled}(\overline{x})$$

where no_transition_enabled is specified as:

$$\mathrm{no\_transition\_enabled}(\overline{x}) \equiv \forall r_i \in \Sigma_{original} : \neg\phi_i(\overline{x})$$

Explicitly defining no_transition_enabled($\overline{x}$) as above and then computing the automaton and intersecting with the fixed-point ξ_{final}, is not so efficient. We have therefore implemented a simple deadlock detector that directly computes ("on the fly") the automaton defining the relation deadlock($\overline{q},\overline{x}$) according to the definition above throughout the construction of ξ_{final}.
A drawback of Boudet-Comon's method is that, from the automaton, there is no known way to derive an explicit expression (like, e.g., a quantifier-free formula) of the characterized arithmetic relation. It is however possible to *enumerate* the set of solutions of the arithmetic relation. This set is usually infinite, but

sometimes one is just interested by knowing the existence and/or the form of one solution (e.g., a path leading to a deadlock marking). Besides, by projection on to an appropriate subset of variables, it is often possible to reduce the infinite space of solutions to a finite one, thus extracting some useful information (e.g., boundedness of some places). See, e.g., section 7.2.

7 Experimental Results

In this section we present some experimental data from three Petri nets having infinite reachability sets: the two first ones have parametrized initial markings while the third one has some unbounded places. We generate for each of them the reachability sets under the form of a Boudet-Comon automaton, and are then able to prove for them various properties. The implementation has been written in SICSTUS-Prolog by the second author. It is around 4000 lines long, and runs on SPARC-10. With each example, we give two tables. The columns of the first table are to be interpreted as: S for 'Stratification', MT for 'Monotonic Transition', MG for 'Monotonic Guard', PoF for 'Cyclic Post-Fusion', PrF for 'Cyclic Pre-Fusion' and ND for 'No Decomposition applies'. The number in each column is the number of times the corresponding decomposition rule was applied. The second table IT (Invariant Transition set) has two rows: The top row is the number of transitions in the set, and the bottom row is the number of times a set of this size was discarded. The explicit form of the flat computed language L will be also given. (Recall that the paths belonging to L "cover" all the reachable markings of the net.)

7.1 Swimming Pool

This example comes from M. Latteux (see, e.g.,[6]). Consider the Petri net in figure 1. With the initial marking $x_1 = x_2 = x_3 = x_4 = x_5 = 0$, $x_6 = q_1$ and $x_7 = q_2$ for some parameters q_1 and q_2, the task is to show that there exists a deadlock whatever the values of q_1 and q_2 are. (The proof is done by hand in [6].) Our implementation does not succeed in computing the fixpoint since the automaton representing the reachability set grows too large (SICSTUS aborts after having generated 2500 states when determinizing an automaton having 386 states with 252 transitions leaving each state). So we apply our method not on the original net, but on a reduced version obtained by applying manually Berthelot's postfusion rule (fusing r_2 and r_3, and *eliminating* x_2) [1]. The reduced net is represented at figure 2. For any values of q_1 and q_2, the reduced net is guaranteed to be deadlock-free iff the original one is.
Computing the parametric reachability set we have the following statistics:

S	MT	MG	PoF	PrF	tot	ND
3	0	2	4	1	10	0

IT:

no. transitions	1	2	3	tot
no. disposals	8	1	3	12

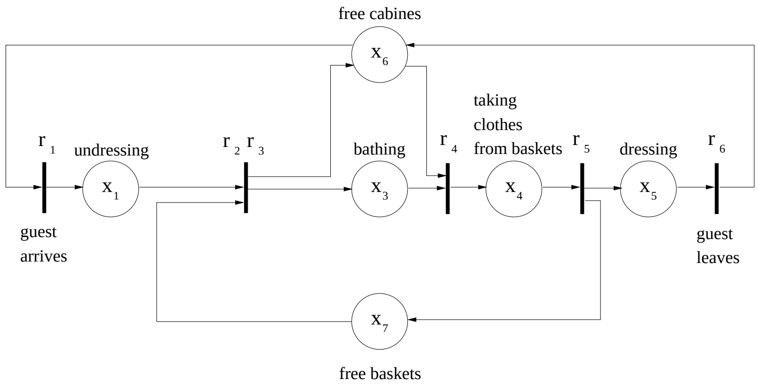

Figure 2

The flat language is computed in 10422 seconds (3.9 hours), and is:
$r_1^*(r_2r_3r_1)^*(r_2r_3)^*r_4^*r_5^*(r_2r_3)^*(r_4r_5r_2r_3)^*(r_4r_5)^*r_1^*r_4^*$
For the reduced swimming pool net of figure 2 the relation deadlock$(q_1, q_2, \overline{x})$ is computed in 12.27 seconds, and: $\forall q_1, q_2\ \exists \overline{x}:$ deadlock$(q_1, q_2, \overline{x})$ (that is, for any q_1 and q_2 there is a deadlock) is verified in 0.02 seconds. For every couple of values c_1 and c_2 for q_1 and q_2, the system can compute path vectors in order to characterize the paths leading to a deadlock. This yields paths of the form $r_1^{c_1}(r_2r_3r_1)^{c_2}$.

7.2 Manufacturing System

This example is taken from [5] (cf. [19]). Consider the Petri net of figure 3. It

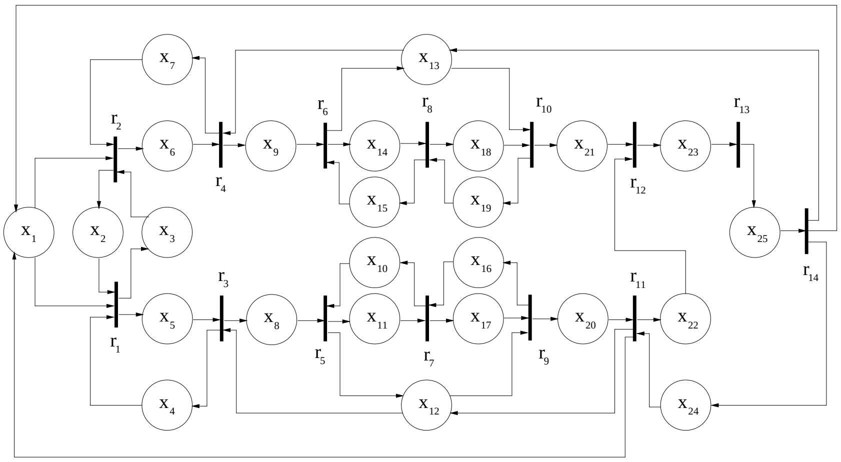
Figure 3

models an automated manufacturing system with four machines, two robots, two buffers (x_{10} and x_{15}) and an assembly cell. The initial marking is: $x_1 = q$ for some nonnegative parameter q, $x_2 = x_4 = x_7 = x_{12} = x_{13} = x_{16} = x_{19} = x_{24} = 1$, $x_{10} = x_{15} = 3$ (thus, the buffers have capacity 3). All other places are empty (that is, all other variables are 0). The task is to discover for which values of q the system may end up in deadlock. (In [19], deadlock-freeness is shown only for $1 \leq q \leq 4$. In [5], deadlock-freeness is proved using some mixed integer programming techniques for $1 \leq q \leq 8$; A path leading to a deadlock is then generated for $q = 9$.) Computing the reachability set, we get the following statistics:

S	MT	MG	PoF	PrF	tot	ND
0	2	61	37	14	114	0

IT:

no. transitions	1	2	3	4	5	6	7	8	9	10	11	12	tot
no. disposals	42	24	18	24	19	13	8	17	2	1	2	1	171

The flat language is computed in 23396 seconds (6.5 hours). It is
$r_1^* r_2^* r_4^* r_6^* r_8^* r_{10}^* r_3^* r_1^* r_2^* r_4^* r_6^* r_{10}^* r_8^* r_5^* (r_3 r_5)^* (r_1 r_3 r_5)^* \; r_2^* r_4^* r_6^* r_{10}^* r_8^* r_1^* r_2^* r_4^* r_6^* r_{10}^* r_8^*$
$r_3^* r_1^* r_2^* r_4^* r_6^* r_{10}^* r_8^* r_7^* r_5^* r_3^* r_1^* r_2^* (r_9 r_7)^* r_9^* r_{11}^* r_3^* r_5^* r_2^* \; r_1^* r_3^* r_5^* r_7^* (r_2 r_1)^* r_2^* r_4^* r_6^* r_{10}^* r_8^*$
$r_{10}^* r_9^* r_7^* r_{12}^* r_{13}^*$.
The relation deadlock$(q, \overline{x})$ is computed in 11.9 seconds. Define the relation:

$$\text{live}(q) \;\equiv\; \neg \exists \overline{x} : \; \text{deadlock}(q, \overline{x})$$

Thus live(q) is the set of parameters for which there is no deadlock in the system. It is computed in 0.09 seconds, and found to be finite in 0.01 seconds. Its enumeration then gives: $\{1, 2, 3, 4, 5, 6, 7, 8\}$. We have therefore a fully automated proof that the system is deadlock free for all the initial markings (of the form given above) for which $1 \leq q \leq 8$, and that for all other value of q, a deadlock exists (note that from deadlock$(q, \overline{x})$, all the deadlock markings for any q may be retrieved, as well as a path to any of them). To prove that the net is bounded for any q amounts to verifying: $\forall q \, \exists \overline{b} \, \forall \overline{x} : \; \xi_{final}(q, \overline{x}) \;\Rightarrow\; \overline{x} \leq \overline{b}$. Our system is too naively implemented to prove this formula as stated, so instead we verify something stronger. We eliminate by projection parameter q and variable x_1 into ξ_{final}, thus getting:

$$\text{subsystem}_1(x_2, x_3, \ldots, x_{25}) \;\equiv\; \exists q, x_1 : \; \xi_{final}(q, x_1, x_2, \ldots, x_{25})$$

The relation subsystem$_1(x_2, x_3, \ldots, x_{25})$ is computed in 38.74 seconds and is shown to be finite (it has 2144 elements) in 1.22 seconds. This shows that all the places but x_1 are bounded. Secondly we compute

$$\text{subsystem}_2(q, x_1) \;\equiv\; \exists x_2, x_3, \ldots, x_{25} : \; \xi_{final}(q, x_1, x_2, \ldots, x_{25})$$

in 7.89 seconds and prove: subsystem$_2(x_1, q) \;\Rightarrow\; x_1 \leq q$ in 0.03 seconds. Therefore the system is bounded for all values of q.

7.3 Alternating Bit Protocol

This example is taken from [4] where all the correctness proofs are done by hand. Consider the alternating bit protocol of figure 4. The initial marking is:

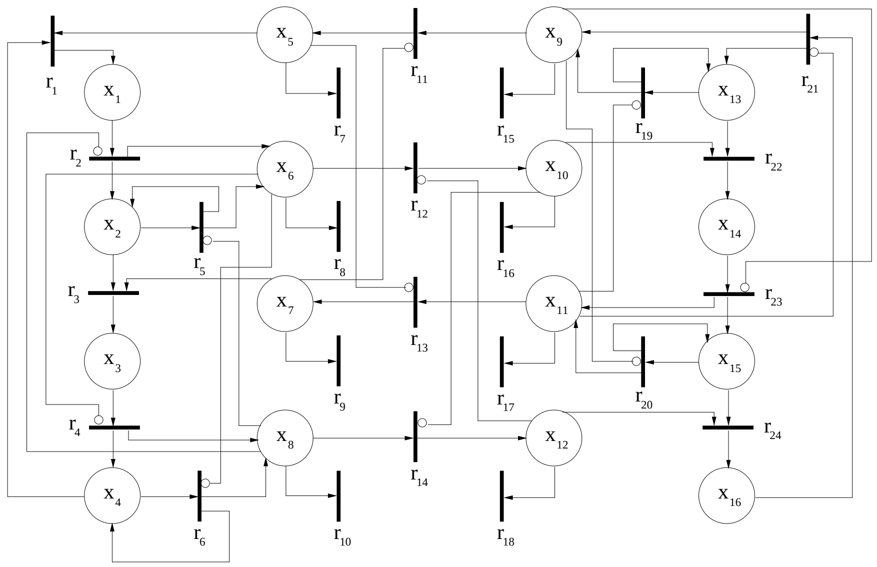

Figure 4

$x_1 = 1, x_{13} = 1$ and $x_i = 0$ for $1 \leq i \leq 14, i \neq 1, i \neq 13$. Note that the system has 8 inhibitor places (which are are the places linked to inhibitor arcs, represented as circle-headed arrows, on the figure). These places are simulated with 8 extra variables. Since there are 16 places in the net, we get a problem with 24 variables. In this example, when computing the reachability set, the decomposition process fails several times. So the system drops some transition chosen according to a simple heuristic (basically random choice). It may then happen that the decomposition process ends without having reached the least fixed-point, in which case the process is restarted with the formula ξ lastly generated as a new base case (see section 5.2). We have conducted the experiment many times with this example, and always eventually reached the least fixed-point. We give below statistics for a a typical computation (which succeeded after three rounds of decomposition).

S	MT	MG	PoF	PrF	tot	ND
12	24	48	26	0	110	6

IT:

no. transitions	1	2	4	5	6	7	8	9	10	11	12	13
no. disposals	23	7	3	4	6	4	9	7	6	3	6	3

The first decomposition round ends after 33 seconds, and yields language

$$L_1 = r_2^* r_8^* r_{12}^* r_{22}^* r_{23}^* r_{20}^* r_{17}^* r_{19}^* r_{22}^* r_{13}^* r_3^* r_4^* r_6^* r_{14}^* r_{10}^* r_{24}^* r_{21}^* r_{19}^* r_{15}^* .$$

At the second round, the decomposition ends after 48 seconds, yielding:
$L_2 = r_{11}^* r_{23}^* r_{20}^* r_{17}^* r_1^* r_2^* r_5^* r_3^* r_8^*$.
At the third round, the decomposition ends after 47 seconds, yielding:
$L_3 = r_{12}^* r_4^* r_6^* r_{10}^*$.
for which the least fixed-point is reached. The flat language for this example is therefore $L_1 L_2 L_3$.
The correctness of the protocol is expressed as follows (see [4]):

$$\begin{array}{rl}
\text{i.} & \xi_{final}(\overline{x}) \ \wedge \ x_1 = 1 \ \Rightarrow \ x_{13} = 1 \ \wedge \ x_6 = x_{10} = x_{11} = x_7 = 0 \\
\text{ii.} & \xi_{final}(\overline{x}) \ \wedge \ x_3 = 1 \ \Rightarrow \ x_{15} = 1 \ \wedge \ x_8 = x_{12} = x_9 = x_5 = 0 \\
\text{iii.} & \xi_{final}(\overline{x}) \ \wedge \ x_{14} = 1 \ \Rightarrow \ x_2 = 1 \ \wedge \ x_{11} = x_7 = x_8 = x_{12} = 0 \\
\text{iv.} & \xi_{final}(\overline{x}) \ \wedge \ x_{16} = 1 \ \Rightarrow \ x_4 = 1 \ \wedge \ x_9 = x_5 = x_6 = x_{10} = 0
\end{array}$$

These four implications were proved in 1.52 seconds each. In 8 seconds, the unbounded places x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} and x_{12} were found. (Note that they coincide exactly with the inhibitor places.)

8 Final Remarks

We have illustrated on three nontrivial examples of Petri nets how our top-down method of decomposition enhanced by forward propagation of the initial values and invariance checks, allows us to characterize arithmetically infinite reachability sets and (dis)prove automatically various safety properties. We have also successfully applied our procedure to examples of automata with counters taken from [10], and on classical examples with finite reachability sets such as dining-philosophers or Peterson's mutual exclusion algorithm. As observed by Hiraishi [11], such a kind of method is not universal because it is known that there exist Petri nets whose reachability sets are not characterizable in Presburger arithmetic. However in practice, the main problem that we have to deal with is the state explosion problem, which prevents the construction of Boudet-Comon's automaton. We have indicated one way to alleviate this problem by reducing the original net to a simpler one that retains its main safety properties (through Berthelot's transformations). Another way that we would like to explore is to use a *compositional* approach, in order to reduce the verification of a global safety property to the verification of several local ones (see, e.g., [7, 15, 17]).

References

1. G. Berthelot. "Transformations and Decompositions of Nets". *Advances in Petri Nets*, LNCS 254, Springer-Verlag, 1986, pp. 359-376.
2. A. Boudet and H. Comon, "Diophantine Equations, Presburger Arithmetic and Finite Automata", *Proc. CAAP*, LNCS 1059, Springer-Verlag, 1996, pp. 30–43.
3. G.W. Brams. *Réseaux de Petri: Théorie et Pratique*, Masson, Paris, 1983.
4. J.M. Couvreur and E. Paviot-Adet. "New Structural Invariants for Petri Nets Analysis". *Proc. Application and Theory of Petri Nets*, LNCS 815, Springer-Verlag, 1994.

5. F. Chu and X. Xie. *Deadlock Analysis of Petri Nets Using Siphons and Mathematical Programming.* Submitted to *IEEE Trans. on Robotics and Automation*, 1996, 29 pages.
6. R. David and H. Alla. *Du Grafcet aux Réseaux de Petri*, Hermès, Paris, 1989.
7. A. Finkel and L. Petrucci. "Composition/Décomposition de Réseaux de Petri et de leurs Graphes de Couverture". *Informatique Théorique et Applications 28:2*, 1994, pp. 73-124.
8. L. Fribourg and H. Olsén. *A Decompositional Approach for Computing the Least Fixed-points of Datalog Programs with Z-counters.* Technical Report LIENS-96-12, Ecole Normale Supérieure, Paris, July 1996. (Available on http://www.dmi.ens.fr/dmi/preprints)
9. S. Ginsburg. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, 1966.
10. N. Halbwachs. "Delay Analysis in Synchronous Programs", *Proc. Computer Aided Verification*, LNCS 697, Springer-Verlag, 1993, pp. 333-346.
11. K. Hiraishi. "Reduced State Space Representation for Unbounded Vector State Spaces", *Proc. Application and Theory of Petri Nets*, LNCS 1091, Springer-Verlag, 1996, pp. 230-248.
12. J. Jaffar and J.L. Lassez. "Constraint Logic Programming", *Proc. 14th ACM Symp. on Principles of Programming Languages*, 1987, pp. 111-119.
13. P. Kanellakis, G. Kuper and P. Revesz. "Constraint Query Languages". Internal Report, November 1990. (Short version in *Proc. 9th ACM Symp. on Principles of Database Systems*, Nashville, 1990, pp. 299-313).
14. R.M. Karp and R.E. Miller. "Parallel Program Schemata". *J. Computer and System Sciences: 3*, 1969, pp. 147-195.
15. F. Laroussinie and K. Larsen. "Compositional Model Checking of Real Time Systems". *Proc. CONCUR*, LNCS 962, Springer-Verlag, 1995, pp. 27–41.
16. M. Presburger. "Uber die Vollstandingen einer gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervotritt", *Comptes Rendus du premier Congrès des Mathématiciens des Pays Slaves*, Varszawa, 1929.
17. A. Valmari. "Compositional State Space Generation", *Advances in Petri Nets*, LNCS 674, Springer-Verlag, 1993, pp. 427-457.
18. H-C. Yen. "On the Regularity of Petri Net Languages". *Information and Computation 124*, 1996, pp. 168-181.
19. M.C. Zhou, F. Dicesare and A.A. Desrochers. "A Hybrid Methodology for Synthesis of Petri Net Models for Manufacturing Systems", *IEEE Trans. on Robotics and Automation*, vol. 8:3, 1992, pp. 350-361.

This article was processed using the LaTeX macro package with LLNCS style