On the Complexity of the linear-time μ-calculus for Petri Nets

Peter Habermehl

VERIMAG
Centre Equation
2, avenue de Vignate, 38610 Gières, France.
email: Peter.Habermehl@imag.fr

Abstract

We study the complexity of model-checking Petri Nets w.r.t. the propositional linear-time μ-calculus. Esparza has shown in [5] that it is decidable, but the space complexity of his algorithm is exponential in the size of the system and double exponential in the size of the formula. In this paper we show that the complexity in the size of the formula can be reduced to polynomial space. We also prove that this is the best one can do. We also show that for the subclass of BPPs the problem has already the same complexity as for arbitrary nets. Furthermore we obtain the same results for the linear time temporal logic LTL, which is strictly less expressive than the linear-time μ-calculus.

1 Introduction

There is now a well established theory for automatic verification of finite state concurrent systems. Recently, the problem of extending this theory to infinite state systems has been addressed. There are mainly two different directions for the automatic verification of infinite state systems. First, the decidability of equivalence relations. Second, the model-checking problem of infinite state systems w.r.t. different temporal logics (branching time as well as linear time). An overview of the first line of research can be found in [9] and for the second in [6]. This paper is situated within the second approach. Various results concerning the decidability of model checking infinite state systems w.r.t. temporal logics have been established the last few years. If we consider infinite state systems which have the power of a Turing machine, then all "interesting" model checking problems are undecidable. Therefore, more restricted systems as Pushdown automata, Petri Nets, Lossy Channel Systems, etc. have been studied. For these systems some temporal logics have been shown to be decidable [1] [6].

The infinite state systems we consider in this paper are Petri Nets and a subclass, Basic Parallel Processes (BPP). BPP's have been introduced by Christensen [3] and provide a basic model of concurrency without synchronization. Many results about them have been proven [5] [7]. Here we consider interleaving and linear semantics of the systems, i.e. systems define sets of infinite transition sequences. The logics we use are linear temporal logics: the well-known propositional linear-time μ-calculus and the less expressive linear temporal logic (LTL

[11]). Logical formulas are interpreted over infinite sequences of atomic propositions. A system satisfies a temporal logic formula iff all the infinite sequences given by the system satisfy the formula. The satisfaction problem of a formula is known as the model-checking problem. Model-checking finite-state systems w.r.t linear-time μ-calculus is in polynomial space in the size of the formula, whereas it is in linear time in the size of the system.

Esparza [5] has studied the model-checking problem of Petri Nets and BPP's w.r.t. various temporal logics. He has shown among other things that model checking Petri Nets w.r.t. the linear-time μ-calculus is decidable. The space complexity of his algorithm is exponential in the size of the Petri Net (the number of places) and double exponential in the size of the formula. In this paper we show how to reduce the space complexity to polynomial space in the size of the formula, which is the same as for finite-state systems. Furthermore we prove that our upper bound in the size of the system is almost optimal by showing that model-checking Petri Nets w.r.t. linear-time μ-calculus is EXPSPACE-hard in the size of the system.

We show that this lower bound can already be established for model-checking of the subclass BPP. This is quite surprising because BPP is a rather weak formalism compared to Petri Nets. On the other hand we show that all these results also hold if we consider the linear temporal logic LTL, which is strictly less expressive than the linear-time μ-calculus.

To obtain an upper bound of the space complexity for the different model-checking problems, we use Vector Addition Systems with States (VASS), which are essentially Petri Nets with control states. The model-checking problem is transformed into the *control-state repeating problem* in VASS, i.e. the problem if there exists an infinite transition sequence which passes infinitely often through some fixed control state. We analyze the complexity of this problem in VASS by using the techniques of [12] and [13] who prove upper bounds for the boundedness problem of Petri Nets. The lower bound is obtained by a reduction from an EXPSPACE-hard problem in Petri Nets.

The paper is organized as follows: In the next chapter we give the basic definitions. Then in chapter 3 we prove a complexity result for the control state repeating problem in VASS. This result is used in chapter 4 to prove an upper bound for the model-checking problem. Chapter 5 gives a lower bound and finally we conclude.

2 Basic definitions and results

In this section we give the basic definitions and lemmas concerning sequences, languages, VASS, Petri Nets, BPP's, ω-automata and linear time temporal logics.

2.1 Sequences, Languages

Let Σ be a finite alphabet. We denote by Σ^* (resp. Σ^ω) the set of finite (resp. infinite) sequences over Σ. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. A *language* (resp. *$\omega$-language*) is a subset of Σ^* (resp. Σ^ω). An infinite sequence $\sigma \in \Sigma^\omega$ can be seen as a mapping from $\mathbb{N}$ to Σ. Hence, σ is equal to $\sigma(0)\sigma(1)\cdots$. We denote by σ^i the infinite sequence $\sigma(i)\sigma(i+1)\cdots$.

2.2 VASS and Petri Nets

Let $\mathbb{Z}$, (resp. $\mathbb{N}$, $\mathbb{N}^+$) denote the integers (resp. nonnegative, positive integers). For some vector $\vec{u} \in \mathbb{Z}^k$ and an i with $1 \leq i \leq k$, the i-th component of $\vec{u}$ (also called i-th place) is denoted by $\vec{u}(i)$. Let $\vec{u}$,$\vec{v} \in \mathbb{Z}^k$. Then, the addition $\vec{u} + \vec{v}$, the subtraction $\vec{u} - \vec{v}$ and the predicates $\vec{u} = \vec{v}$, $\vec{u} \geq \vec{v}$ and $\vec{u} < \vec{v}$ are defined as usual. $\vec{0}$ denotes the zero vector.

A k-dimensional labelled *vector addition system with states* (VASS) is a 6-tuple $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$, where

- Σ is a finite *alphabet*,
- $\vec{v}_0 \in \mathbb{N}^k$ is called the *start vector*,
- A is a set of vectors in $\mathbb{Z}^k$ (the *addition set*),
- Q is a finite set of *control* states,
- $q_0 \in Q$ is the *initial* state,
- $\delta \subseteq Q \times \Sigma \times Q \times A$ is the *transition relation*.

A *transition* $(q_1, b, q_2, \vec{a}) \in \delta$ is usually written in the form $q_1 \xrightarrow{b} (q_2, \vec{a})$ (if we do not care about b we also write $q_1 \rightarrow (q_2, \vec{a})$). A *configuration* of the VASS is a tuple $(q, \vec{v})$, where $q \in Q$ and $\vec{v} \in \mathbb{N}^k$. The *initial configuration* is the tuple $(q_0, \vec{v}_0)$. A configuration $(q_1, \vec{v})$ can perform a transition $q_1 \xrightarrow{b} (q_2, \vec{a})$ giving the configuration $(q_2, \vec{v} + \vec{a})$ provided that $\vec{v} + \vec{a} \geq \vec{0}$. This is denoted by $(q_1, \vec{v}) \overset{b}{\rightsquigarrow} (q_2, \vec{v} + \vec{a})$ (or $(q_1, \vec{v}) \rightsquigarrow (q_2, \vec{v} + \vec{a})$). This definition is generalized in the obvious way to sequences in Σ^∞. By $\rightsquigarrow^+$ (resp. $\rightsquigarrow^*$), we denote the transitive (resp. transitive and reflexive) closure of $\rightsquigarrow$.

The *ω-language* of a VASS $\mathcal{S}$, called $L(\mathcal{S})$, is the set of infinite sequences $\sigma \in \Sigma^\omega$ such that $(q_0, \vec{v}_0) \overset{\sigma}{\rightsquigarrow}$.

A labelled Petri Net $\mathcal{N}$ is a VASS $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ with one state, i.e $Q = \{q_0\}$. It is easy to see that for each k-dimensional VASS with n states there exists an equivalent Petri Net with $k + n$ dimensions (each state corresponds to one new place). In this paper we use VASS to have a convenient representation of the product between Petri Nets and Büchi automata.

2.3 Basic Parallel Processes (BPP)

A Basic Parallel Processes [3] is defined by a finite set Δ of well-guarded recursive equations of the form $X = t$ where X is a process variable and t is a term constructed from a finite set of transition labels Σ, process variables, and binary operators: nondeterministic choice "+", prefixing "$b \cdot X$" and merge (or asynchronous parallel) composition "$\|$". One variable is singled out as the leading variable. The computation has to start from there. An operational semantics associates with each BPP process a labelled transition system (infinite in general). The transition relations $\xrightarrow{b}$ are defined to be the least relations satisfying:

- $b \cdot X \xrightarrow{b} X$,
- if $X = t \in \Delta$ and $t \xrightarrow{b} t'$ then $X \xrightarrow{b} t'$,
- if $t_1 \xrightarrow{b} t_1'$ then $t_1 + t_2 \xrightarrow{b} t_1'$,
- if $t_2 \xrightarrow{b} t_2'$ then $t_1 + t_2 \xrightarrow{b} t_2'$,
- if $t_1 \xrightarrow{b} t_1'$ then $t_1 \| t_2 \xrightarrow{b} t_1' \| t_2$,
- if $t_2 \xrightarrow{b} t_2'$ then $t_1 \| t_2 \xrightarrow{b} t_1 \| t_2'$

The ω-language $L(\mathcal{S}) \subseteq \Sigma^\omega$ of a BPP $\mathcal{S}$ is the set of infinite sequences starting from the leading variable.

BPP's correspond to a subclass of VASS called BPP-Nets.

A BPP-Net is a VASS $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ such that $Q = \{q_0\}$ and A contains only vectors which have at most one negative component. This component must have the value -1.

We can show that BPP-Nets and BPP Processes define the same class of labelled transition systems. To translate a BPP Process $\mathcal{P}$ into a BPP-Net we construct an equivalent process in guarded normal form [3] for $\mathcal{P}$ where all right-hand sides of the equations are of the form $\sum_{i=1}^n b_i \cdot (X_{i_1} \| \cdots \| X_{i_m})$. By renaming variables we can assume that the process variable on the left-hand side of an equation does not appear on the right-hand side of the same equation. Then we associate with each process variable a component of a k-dimensional vector (k is the number of variables). Each equation $X = \sum_{i=1}^n b_i \cdot (X_{i_1} \| \cdots \| X_{i_m})$ is translated to n transitions. Every term $b_i \cdot (X_{i_1} \| \cdots \| X_{i_m})$ corresponds to a transition $q_0 \xrightarrow{b_i} (q_0, \vec{a})$ where the component in the vector $\vec{a}$ representing X is -1 and the other components indicate the number of times a process variable appears in $b_i \cdot (X_{i_1} \| \cdots \| X_{i_m})$. The translation in the other direction is also easily done. Basic Parallel Processes are a rather weak model compared to Petri Nets because there is no synchronization allowed (each transition has at most one input place).

2.4 ω-automata

A finite-state ω-automaton is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, C)$ where (Σ, Q, q_0, δ) is a finite-state labelled transition system (LTS), let us call it $\mathcal{S}_\mathcal{A}$, and C a set of acceptance conditions. Given a sequence $\sigma \in \Sigma^\omega$, a run of $\mathcal{S}_\mathcal{A}$ over σ is a sequence $\rho \in Q^\omega$ such that $\rho(0) = q_0$, and $\forall i \geq 0$, $(\rho(i), \sigma(i), \rho(i+1)) \in \delta$. Let $\rho \in Q^\omega$ be a run of $\mathcal{S}_\mathcal{A}$. We denote by $Inf(\rho)$ the set of locations q such that $\exists^\infty i \in \mathbb{N}$ with $\rho(i) = q$.

Then, a run ρ is *accepting* if it satisfies the acceptance conditions. The acceptance conditions we use are summarized in the following table:

Type	Syntax	Semantics
Büchi	$F \subseteq Q$	$Inf(\rho) \cap F \neq \emptyset$
Rabin	$\bigvee_i L_i \wedge \neg U_i$	$\exists i : Inf(\rho) \cap L_i \neq \emptyset \wedge Inf(\rho) \cap U_i = \emptyset$
Streett	$\bigwedge_i L_i \to U_i$	$\forall i : Inf(\rho) \cap L_i = \emptyset \vee Inf(\rho) \cap U_i \neq \emptyset$

where L_i and U_i (*accepting pair*) are subsets of Q. The ω-*language* of $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of sequences $\sigma \in \Sigma^\omega$ such that $\mathcal{S}_\mathcal{A}$ has an accepting run over σ. Two ω-automata are called *equivalent* iff they have the same ω-language.

In section 2.5 we construct a Büchi automaton $\mathcal{A}_\varphi$ equivalent to a formula φ in the linear-time μ-calculus. The construction involves complementation and intersection of Büchi automata. For the proof of the lower complexity bound for the model-checking problem in section 4 we have to construct $\mathcal{A}_\varphi$ "on-the-fly", i.e. we calculate the information about the automaton only when needed in our model-checking algorithm. Therefore we require bounds on the size of a description of one state and on the size of the automaton in these constructions. The following lemmas provide these bounds.

Lemma 2.1 *For any two Büchi automata $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) with n_1 (resp. n_2) states where the description of one state has size d_1 (resp. d_2), there exists a Büchi automaton $\mathcal{A}_3$ with $2n_1n_2$ states such that $L(\mathcal{A}_3) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. During the construction of $\mathcal{A}_3$ the description of a state has size $d_1 + d_2 + 1$.*

This follows from the usual product construction for Büchi automata (see for example [18]).

Lemma 2.2 *For any Büchi automaton $\mathcal{A}_1$ with n states, there exists an equivalent deterministic Rabin automaton $\mathcal{A}_2$ with $2^{O(nlog(n))}$ states and $O(n)$ accepting pairs. During the construction of $\mathcal{A}_2$ the description of a state has size $O(n^2)$.*

This follows from Safra's determinization procedure [14]. The size of a description of a state of $\mathcal{A}_2$ is $O(n^2)$ because a state is a tree with at most n nodes and every node is a subset of the set of states of $\mathcal{A}_1$. Notice that each accepting pair is uniquely determined by one node of the tree.

Lemma 2.3 *For any Streett automaton $\mathcal{A}_1$ with n states (with a description of one state of size d) and h accepting pairs, there exists an equivalent Büchi automaton $\mathcal{A}_2$ with $n \cdot 2^{O(h)}$ states. During the construction of $\mathcal{A}_2$ the description of a state has size $d \cdot O(h^2)$.*

Also follows from [14].

Lemma 2.4 *For any Büchi automaton $\mathcal{A}_1$ with n states, there exists a complementary Büchi automaton $\mathcal{A}_2$ with $2^{O(nlog(n))}$ states. During the construction of $\mathcal{A}_2$ the description of a state has size $O(n^4)$.*

To obtain $\mathcal{A}_2$ we first determinize with lemma 2.2. Then we obtain the complement by completing the Rabin automaton and interpreting it as a Streett automaton. Finally, we obtain a Büchi automaton with required size by lemma 2.3.

2.5 Linear time temporal Logics

The linear time μ-calculus Let $\mathcal{X}$ be a set of variables. Then a formula of the propositional linear time μ-calculus is given by the following syntax:

$$\varphi ::= \; Z \in \mathcal{X} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \mu Z.\varphi(Z)$$

with a monotonicity condition: Given a formula $\mu Z.\varphi(Z)$, every free occurrence of Z in $\varphi(Z)$ must be in the scope of an even number of negations. We use the usual abbreviations, in particular $\nu Z.\varphi(Z) = \neg\mu Z.\neg\varphi(\neg Z)$, the greatest fixpoint operator.

A *model* $\mathcal{M}$ of a formula assigns to each variable $Z \in \mathcal{X}$ a subset $\mathcal{M}(Z) \subseteq I\!N$. Models are extended to arbitrary formulas with free variables in the following way:

$$\begin{aligned}
\mathcal{M}(\neg\varphi) &= I\!N \setminus \mathcal{M}(\varphi)\\
\mathcal{M}(\varphi \wedge \psi) &= \mathcal{M}(\varphi) \cap \mathcal{M}(\psi)\\
\mathcal{M}(\bigcirc\varphi) &= \{i \mid i+1 \in \mathcal{M}(\varphi)\}\\
\mathcal{M}(\mu Z.\varphi(Z)) &= \bigcap\{A \subseteq I\!N \mid \mathcal{M}[Z \mapsto A](\varphi) \subseteq A\}
\end{aligned}$$

where $\mathcal{M}[Z \mapsto A]$ is the model which assigns A to Z and agrees with $\mathcal{M}$ on the other variables. Let $\mathcal{Y} \subseteq \mathcal{X}$ be the set of free propositional variables in a formula φ and $\Sigma = 2^{\mathcal{Y}}$. Then a model $\mathcal{M}$ of φ determines one infinite sequence $\sigma_{\mathcal{M}} \in \Sigma^\omega$ given by $\sigma_{\mathcal{M}}(i) = \{Z \in \Sigma \mid i \in \mathcal{M}(Z)\}$. The *interpretation* of φ (denoted $[\![\varphi]\!]$) is defined by $[\![\varphi]\!] = \{\sigma_{\mathcal{M}} \mid 0 \in \mathcal{M}(\varphi)$ for some model $\mathcal{M}\}$, e.g the set of all infinite sequences satisfying φ. We have the following well-known theorem on the relation between μ-calculus and Büchi automata.

Theorem 2.1 *Given a formula in the linear-time μ-calculus φ, there is a Büchi automaton $\mathcal{A}_\varphi$, such that $L(\mathcal{A}_\varphi) = [\![\varphi]\!]$ and vice-versa.*

For a translation from Büchi automaton to linear-time μ-calculus see for example [10] or [4]. The construction of the automaton from a given formula is detailed in [16]. Here we give the broad outline of the construction, which is needed to prove an upper bound for our model-checking problem.

Let n be the size of φ. To construct the corresponding Büchi automaton we first of all transform φ into a formula φ' in positive normal form, i.e. all subformulas which are not propositional variables are positive. This can be done with a linear increase in size by using standard transformation rules as $\neg\mu Z.\varphi(Z) = \nu Z.\neg\varphi(\neg Z)$.

Then $[\![\varphi']\!]$ is given as the projection on the free propositional variables of the intersection of two Büchi automata $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}_1$ corresponds to the formula without taking into account fixpoints, i.e. it accepts the so-called pre-models of the formula. $\mathcal{A}_1$ accepts sequences which are not models, because it does not prevent evaluations of least fixpoints not to terminate. Thus $\mathcal{A}_2$ is the complement of the Büchi automaton $\mathcal{A}_3$ which gives all the sequences where some least fixpoint is infinitely often regenerated (see [16] for details of the construction). $\mathcal{A}_3$ has

$O(n^2)$ states. Now, by using lemma 2.4 we see that $\mathcal{A}_2$ has $2^{O(n^2 log(n))}$ states and a description of a state during the construction of $\mathcal{A}_2$ has polynomial size. Thus with lemma 2.1 we obtain

Lemma 2.5 *Given a formula in the linear-time μ-calculus φ of size n, there is a Büchi automaton $\mathcal{A}_\varphi$ with $2^{O(n^2 log(n))}$ states such that $L(\mathcal{A}_\varphi) = [\![\varphi]\!]$. Furthermore, during the construction of $\mathcal{A}_\varphi$ the description of a state has polynomial size in n.*

Linear-time temporal logic (LTL) Let $\mathcal{X}$ be a set of propositional variables. Then a formula of the *linear time temporal logic* (LTL) is given by the syntax

$$\varphi ::= Z \in \mathcal{X} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi \mathcal{U} \varphi$$

A *model* $\mathcal{M}$ of a formula assigns to each variable $Z \in \mathcal{X}$ a subset $\mathcal{M}(Z) \subseteq I\!N$. Models are extended to arbitrary formulas with free variables in the following way:

$$\begin{aligned}
\mathcal{M}(\neg\varphi) &= I\!N \setminus \mathcal{M}(\varphi) \\
\mathcal{M}(\varphi \wedge \psi) &= \mathcal{M}(\varphi) \cap \mathcal{M}(\psi) \\
\mathcal{M}(\bigcirc\varphi) &= \{i \mid i+1 \in \mathcal{M}(\varphi)\} \\
\mathcal{M}(\varphi \mathcal{U} \psi) &= \{i \mid \exists j \geq i \,.\, \forall\, k \text{ with } i \leq k < j.\ k \in \mathcal{M}(\varphi) \text{ and } j \in \mathcal{M}(\psi)\}
\end{aligned}$$

We use the usual abbreviations: $\Diamond\varphi = true\mathcal{U}\varphi$ and $\Box\varphi = \neg\Diamond\neg\varphi$. Then, in the same way as in section 2.5 for the linear time μ-calculus we define the infinite sequences in Σ^ω given by a model and the interpretation $[\![\varphi]\!]$ of a formula φ.

It is well known that LTL is strictly less expressive than the linear time μ-calculus, i.e. for every formula φ in LTL, there exists a formula φ' in the μ-calculus such that $[\![\varphi]\!] = [\![\varphi']\!]$ and the contrary does not hold. For example the fact that at each even point of every computation sequence some propositional property holds can be expressed in the μ-calculus but not in LTL [19].

3 Complexity results for VASS

We solve the model-checking for Petri Nets and BPP's w.r.t. the linear-time μ-calculus by transforming it to the *control state repeating problem* in VASS, i.e the problem, if there is an infinite transition sequence starting from the initial configuration which visits infinitely often a given control state. In this section we will analyze the space complexity of this problem in detail. We give an upper bound which is parameterized by the dimension of the VASS, the number of states and the biggest absolute value of components of vectors in the addition set. The proof technique we use is similar to the techniques in [12] and [13] which show upper bounds for the boundedness problem of Petri Nets.

3.1 Notation

To be able to prove the upper bound, we need some more definitions. All these definitions and notations are essentially from [12] and [13]. Let $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ be a VASS of dimension k. A *path* of length m starting from $(q, \vec{v})$ is a sequence of pairs $(q_1, \vec{w}_1)$,$(q_2, \vec{w}_2)$, ..., $(q_m, \vec{w}_m)$ such that $(q_1, \vec{w}_1) = (q, \vec{v})$ and $\forall i$, $1 \le i \le m$ $q_i \in Q$,$\vec{w}_i \in \mathbb{Z}^k$ and $\forall i$, $1 \le i < m$, $q_i \to (q_{i+1}, \vec{w}_{i+1} - \vec{w}_i) \in \delta$. Informally, paths are transition sequences which do not respect the property that every vector occuring should be nonnegatif. The *effect* of a path is the vector $\vec{w}_m - \vec{w}_1$.

Let $\vec{w} \in \mathbb{Z}^k$ and $0 \le i \le k$. The vector $\vec{w}$ is called *i bounded* iff $\forall j, 1 \le j \le i, \vec{w}(j) \ge 0$. For a $r \in \mathbb{N}^+$ such that $0 \le \vec{w}(j) < r$ for $1 \le j \le i$, $\vec{w}$ is called *i-r bounded*. A path in a VASS is called *i bounded* (resp. *i-r bounded*) if all the vectors appearing in it are i bounded (resp. i-r bounded).

Remark that for a k bounded path $(q_1, \vec{w}_1), (q_2, \vec{w}_2), \ldots, (q_m, \vec{w}_m)$ of a *k-dimensional* VASS starting from the initial configuration we have $(q_1, \vec{w}_1) \rightsquigarrow (q_2, \vec{w}_2) \rightsquigarrow \ldots \rightsquigarrow (q_m, \vec{w}_m)$.

An i loop is a path $(q_1, \vec{w}_1), \ldots, (q_m, \vec{w}_m)$ where $q_1 = q_m$ and $\vec{w}_1(j) = \vec{w}_m(j)$ for all j,$1 \le j \le i$.

Now, we are able to prove our main result about VASS.

3.2 Control state repeating problem

In this section we obtain an upper bound of the space complexity of the following problem.

Problem 3.1 *Given a VASS $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$, does there exist a transition sequence starting from the initial configuration which passes infinitely often through some fixed state $q \in Q$?*

Because $\le$ on $\mathbb{N}^k$ is a well-ordering (there is no infinite sequence $\vec{v}_1, \vec{v}_2, \ldots$ with $\vec{v}_i \not\le \vec{v}_j$ for all $i < j$) problem 3.1 is easily shown to be equivalent to the problem

Problem 3.2 *Given a VASS $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ and a control state $q \in Q$, do there exist configurations $(q, \vec{w})$, $(q, \vec{w}')$ such that $(q_0, \vec{v}_0) \rightsquigarrow^* (q, \vec{w}) \rightsquigarrow^+ (q, \vec{w}')$ and $\vec{w}' \ge \vec{w}$.*

This problem is similar to the boundedness problem (see [12]) for Petri Nets. Hence, to obtain an upper bound we adapt the corresponding proof of an upper bound for the boundedness problem in [12]. Here we want to obtain an upper bound parameterized by the number of states in the VASS. Thus we follow the approach of Rozier and Yen [13] who analyze the boundedness problem of Petri Nets in more detail.

We are interested in the space-complexity of problem 3.2. Hence we will show that if there is a transition sequence satisfying the property in problem 3.2 then there is a transition sequence of some bounded length satisfying the property. Then, it suffices to check all the sequences up to this length for the property. We prove the upper bound on the length by induction on the dimension of the

VASS. We will show an upper bound for paths which are "correct" (in the sense of $\rightsquigarrow$) for the first i dimensions, i.e. i bounded. Finally we obtain a bound for k-bounded paths which correspond to transition sequences.

To get started let us fix a k-dimensional n-state VASS $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ such that the absolute values of components of vectors in A are smaller than l and a $q \in Q$. A *q-repeating* path starting from $(q', \vec{v})$ in this VASS is defined to be a path starting from $(q', \vec{v})$ of the form $(q_1, \vec{w}_1), (q_2, \vec{w}_2), \ldots, (q_m, \vec{w}_m)$ such that $\exists j.\ 1 \leq j < m$ with $q_j = q$, $q_m = q$ and $\vec{w}_m \geq \vec{w}_j$.

Notice, that a k bounded q-repeating path is a transition sequence satisfying the property in problem 3.2.

Let i with $0 \leq i \leq k$. For each pair $(q', \vec{v})$ with $q' \in Q$ and $\vec{v} \in \mathbb{Z}^k$ define $m(i, q', \vec{v})$ as the length of the shortest i bounded q-repeating path starting from $(q', \vec{v})$, if one exists; otherwise let $m(i, q', \vec{v})$ be 0. Then, define $f(i) = max\{m(i, q', \vec{v}) \mid q' \in Q, \vec{v} \in \mathbb{Z}^k\}$. Notice, that this definition does not depend on the initial state and the start vector of the VASS. We will show an upper bound on $f(i)$ by induction. Ultimately, we are only interested in $f(k)$.

In the induction step we need an upper bound on the length of i-r bounded, q-repeating paths in the VASS $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$. To show this upper bound we need a lemma from linear programming. This lemma is from [12] and the proof can be found in [2].

Lemma 3.1 *Let $d_1, d_2 \in \mathbb{N}^+$, let B be a $d_1 \times d_2$ integer matrix and let b be a $d_1 \times 1$ integer matrix. Let $d \geq d_2$ be an upper bound on the absolute values of the integers in B and b. If there exists a vector $\vec{v} \in \mathbb{N}^{d_2}$ which is a solution of $B\vec{v} \geq b$, then for some constant c independent of d, d_1, d_2, there exists a vector $\vec{v} \in \mathbb{N}^{d_2}$ such that $B\vec{v} \geq b$ and $\vec{v}(i) < d^{cd_1}$ for all $i, 1 \leq i \leq d_2$.*

Lemma 3.2 *If there exists an i-r bounded, q-repeating path starting from $(q', \vec{v})$, then there exists an i-r bounded, q-repeating path starting from $(q', \vec{v})$ with length $< (rnl)^{k^c}$, for some constant c independent of r, l, k and n.*

Proof: The proof follows closely the similar proofs of [12] and [13]. Let us call the i-r bounded, q-repeating path starting from $(q', \vec{v})$ $\rho = (q_1, \vec{w}_1), \ldots, (q_m, \vec{w}_m)$. Because ρ is q-repeating we can obtain two paths ρ_1 and ρ_2 such that $\rho_1 = (q_1, \vec{w}_1), \ldots, (q_{m_0}, \vec{w}_{m_0})$ with $q_{m_0} = q$ and $\rho_2 = (q_{m_0}, \vec{w}_{m_0}) \ldots (q_m, \vec{w}_m)$ where $q_m = q$ and $\vec{w}_m \geq \vec{w}_{m_0}$. We can suppose that ρ_1 is not longer than $r^i n \leq r^k n$, because if not their would be two pairs $(q_{j_1}, \vec{w}_{j_1})$ and $(q_{j_2}, \vec{w}_{j_2})$ with $1 \leq j_1 < j_2 \leq m_0$ such that $q_{j_1} = q_{j_2}$ and $\vec{w}_{j_1}$ and $\vec{w}_{j_2}$ agree on the first i components. And we could obtain a new shorter i-r bounded q-repeating path by "eliminating" from ρ_1 the i loop between $(q_{j_1}, \vec{w}_{j_1})$ and $(q_{j_2}, \vec{w}_{j_2})$. Notice, that this would not change the property $\vec{w}_m \geq \vec{w}_{m_0}$.

To obtain an upper bound on the length of ρ_2 we will essentially show how to eliminate i loops. Notice, that we can not apply the same method as for ρ_1, because we have to make sure that the condition $\vec{w}_m \geq \vec{w}_{m_0}$ still holds after elimination of a loop.

The path ρ_2 can be decomposed into some i loops and the rest of the path ρ_3, such that no pair $(q_j, \vec{w}_j)$ appearing in an i loop does not appear in ρ_3. This

insures that every loop can still be executed after elimination of other loops. The length of ρ_3 is strictly less than $(r^k n+1)^2$. If it is longer, we can always find another i loop, such that no pair in the loop does not appear outside. Now, the length of an i loop can be chosen to be at most $r^i n \leq r^k n$, so loop effects are the sum of at most $r^k n$ vectors which have absolute values of at most l. So, there are at most $(2 \cdot lr^k n+1)^k$ different loop effects. Let us call the different loop effects l_j. Because ρ_2 is an q-repeating path, we know that the system of equations $n_1 l_1 + \cdots + n_p l_p + e(\rho_3) \geq \vec{0}$, where $e(\rho_3)$ is the effect of ρ_3, has a solution in $\mathbb{N}^p$. By letting $d = max(l(r^k n+1)^2, (2 \cdot lr^k n+1)^k)$ and $d_1 = k$ and using lemma 3.1 we see that there is a solution with all the $n_j < (lrn)^{k^c}$ for some constant c. This gives us a bound on the number of times a certain loop has to be executed. Because of the construction of ρ_3 every loop where $n_j > 0$ can be executed starting from a pair in ρ_3. Therefore we can construct from ρ_3 an i-r bounded q-repeating path with length $< (lrn)^{k^c} r^k n + (r^k n+1)^2 + r^k n < (lrn)^{k^{c'}}$.
□

By carefully observing the construction we could obtain a slightly lower bound. This is not done here for sake of clarity. Now we are able to prove an upper bound of $f(i)$ by induction.

Lemma 3.3 *$f(0) < (l^2 n)^{k^c}$ for some constant c independent of l, k and n.*

Proof: Follows from lemma 3.2 by observing the fact, that a 0 bounded q-repeating path is trivially a 0-1 bounded q-repeating path.

Lemma 3.4 *$f(i+1) < (l^2 n f(i))^{k^c}$ for some constant c independent of l, k and n.*

Proof: Consider an arbitrary $(i+1)$ bounded q-repeating path $\rho = (q_1, \vec{w}_1) \ldots (q_m, \vec{w}_m)$ starting from $(q', \vec{v})$.

Case 1: If ρ is $(i+1)$-$lf(i)$ bounded then with lemma 3.2 there is a q-repeating path with length $< (l^2 n f(i))^{k^c}$.

Case 2: ρ is not $(i+1)$-$lf(i)$ bounded. Then, first of all we can construct easily a path $\rho' = \rho\rho_1$ such that $(q_m, \vec{w}_m)\rho_1$ is $(i+1)$ bounded and q-repeating. Now, ρ is not $(i+1)$-$lf(i)$ bounded. So there is a first point m_0 on the path, there the vector is not $lf(i)$ bounded. Without loss of generality we suppose that $\vec{w}_{m_0}(i+1) \geq lf(i)$ and $(q_{m_0}, \vec{w}_{m_0}) \ldots (q_m, \vec{w}_m)\rho_1$ (called ρ_2) is an $(i+1)$ bounded q-repeating path starting from $(q_{m_0}, \vec{w}_{m_0})$.

Furthermore, we can assume that $m_0 < (lf(i))^{i+1} n$, because if not their would be two pairs $(q_{j_1}, \vec{w}_{j_1})$ and $(q_{j_2}, \vec{w}_{j_2})$ with $1 \leq j_1 < j_2 \leq m_0$ such that $q_{j_1} = q_{j_2}$ and $\vec{w}_{j_1}$ and $\vec{w}_{j_2}$ agree on the first i+1 components. And we could obtain a new q-repeating path by "eliminating" from ρ the loop between $(q_{j_1}, \vec{w}_{j_1})$ and $(q_{j_2}, \vec{w}_{j_2})$.

Since ρ_2 is a $(i+1)$ bounded q-repeating path it is trivially an i bounded q-repeating path. So by applying the induction hypothesis, there is an i bounded q-repeating path starting from $(q_{m_0}, \vec{w}_{m_0})$ with length $\leq f(i)$. Now, because

$\vec{w}_{m_0}(i+1) \geq lf(i)$ and each rule can remove at most l of each component the path is also $i+1$ bounded.

Putting together the two paths we obtain an $i+1$ bounded q-repeating path of length $< (lf(i))^{i+1}n + f(i) < (l^2 n f(i))^{k^c}$. □

Now, by using lemmas 3.3 and 3.4 we see that $f(k) < (ln)^{(k^c)^k} = (ln)^{2^{cklog(k)}}$ for some constant c. So a non-deterministic algorithm to solve Problem 3.1 will just "guess" a q-repeating path starting from the initial configuration up to a certain length. This can be done in space $O((log(l) + log(n))2^{cklog(k)})$.

Theorem 3.1 *For a k-dimensional n-state VASS $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$, where the absolute values of components of vectors in A are smaller than l, Problem 3.1 is solvable in $O((log(l)+log(n))2^{cklog(k)})$ nondeterministic space for some constant c independent of l,k and n.*

We use this theorem in the next section to prove an upper bound on the model-checking of Petri Nets w.r.t linear time μ-calculus.

4 The upper bound

In this section we will give an upper bound for the space-complexity of model-checking Petri Nets w.r.t. the propositional linear-time μ-calculus. This problem was proven decidable by Esparza [5]. His definition of the semantics of a formula in the linear-time μ-calculus is different from ours. He considers also finite sequences to be able to express deadlock properties. The model-checking problem in this case is reduced to the reachability problem in Petri Nets, for which no elementary upper bound has been proven so far. A detailed parameterized analysis of the complexity for the reachability problem for VASS has still to be done. However, here we only consider the case that formulas define a set of infinite sequences. For this case, Esparza's algorithm uses exponential space in the size (number of places, dimension) of the Petri Net and double exponential space in the size of the formula. Here we give a more detailed analysis and we can show that the model-checking problem requires only polynomial space in the size of the formula, while needing exponential space in the size of the Petri Net.

The model-checking problem for the linear-time μ-calculus is defined as follows. Given a formula φ interpreted over Σ and a Petri Net $\mathcal{N}$ with alphabet Σ, the Petri Net is said to satisfy the formula φ iff $L(\mathcal{N}) \subseteq [\![\varphi]\!]$. This is equivalent to $L(\mathcal{N}) \cap [\![\neg\varphi]\!] = \emptyset$. To solve this problem, we construct a Büchi automaton $\mathcal{A}_{\neg\varphi}$, which corresponds to $[\![\neg\varphi]\!]$ and then the "product" between this automaton and the Petri Net. We obtain a VASS with acceptance condition. Then we solve the emptiness problem by using the results of section 3.

Let us define the product between a VASS and a Büchi automaton as follows. Given a VASS $\mathcal{S} = (\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ with one state and a Büchi Automaton $\mathcal{A} = (\Sigma, Q', q_0', \delta', F)$. The product $\mathcal{S} \times \mathcal{A}$ is a VASS $(\Sigma, \vec{v}_0, A, Q'', q_0'', \delta'')$ where

– $Q'' = Q \times Q'$,

- $q_0'' = (q_0, q_0')$,
- $((q_1, q_1'), b, (q_2, q_2'), \vec{a}) \in \delta''$ iff $(q_1, b, q_2, \vec{a}) \in \delta$ and $(q_1', b, q_2') \in \delta'$.

Now, obviously $L(\mathcal{S}) \cap L(\mathcal{A}) \subseteq L(\mathcal{S} \times \mathcal{A})$. To obtain the other direction of the set inclusion we have to put an acceptance condition on the runs of $\mathcal{S} \times \mathcal{A}$. Then, the following lemma follows easily.

Lemma 4.1 *Given a VASS $\mathcal{S} = (\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ with one state and a Büchi Automaton $\mathcal{A} = (Q', \Sigma, q_0', \delta', F)$. Then $L(\mathcal{S}) \cap L(\mathcal{A}) \neq \emptyset$ iff there is a transition sequence of $\mathcal{S} \times \mathcal{A} = (\Sigma, \vec{v}_0, A, Q'', q_0'', \delta'')$ starting from the initial state and going infinitely often through some state $q'' = (q, q')$ where $q' \in F$.*

Using this lemma, we see that to solve the model-checking problem for a Petri Net $\mathcal{N}$ which is a VASS $\mathcal{S}$ with one state and a formula φ, we just have to solve some instances of problem 3.1 for the VASS $\mathcal{S} \times \mathcal{A}_{\neg\varphi}$.

In section 3 we gave a space complexity bound of Problem 3.1. The problem is logarithmic in the number of control states of the VASS, because the algorithm which solves the problem non-deterministically guesses the q-repeating paths. We know, that the size of the Büchi automaton $\mathcal{A}_\varphi$ corresponding to some μ-calculus formula φ can be exponential in the length of the formula. If we had to construct the whole automaton, we would need exponential space in the size of the formula for the model-checking problem. But if we construct the automaton "on-the-fly", then we obtain a model-checking algorithm which is polynomial in the size of the formula. "On-the-fly" means that given φ, the algorithm will calculate the information about $\mathcal{A}_{\neg\varphi}$ only then needed in the algorithm for solving Problem 3.1. The algorithm guesses one repeating state r and then tries to non-deterministically construct a sequence from the initial state leading to r and a sequence from r to r. At every step we only have to remember 3 states of the automaton. Theorem 3.1 gives as an upper bound on the number of steps needed. This upper bound can be stored in polynomial space in the size of the formula.

To prove that we can construct $\mathcal{A}_{\neg\varphi}$ on-the-fly we have to show (see [18]) that a description of a state of $\mathcal{A}_{\neg\varphi}$ needs at most polynomial space and that all the information about the automaton (are two state related by δ ?, is a state repeating ?, etc.) can be calculated in polynomial space.

We prove it by carefully examining the construction of $\mathcal{A}_{\neg\varphi}$. Lemma 2.5 shows that a description of a state of $\mathcal{A}_{\neg\varphi}$ has polynomial size. Furthermore all the steps of the construction (determinization,intersection) can be done on-the-fly in polynomial space.

Using the well known theorem of Savitch [15] we can eliminate non-determinism in Theorem 3.1 and obtain the following two theorems. Notice, that the size of the Petri Net is defined to be the number of places (dimensions).

Theorem 4.1 *The model-checking problem of Petri Nets w.r.t. to the linear time μ-calculus can be solved in deterministic PSPACE in the size of the formula.*

Theorem 4.2 *The model-checking problem of Petri Nets w.r.t. to the linear time μ-calculus can be solved in $O(2^{cnlog(n)})$ deterministic space for some constant c where n is the size of the Petri Net.*

Because BPP's are strictly less expressive than Petri Nets, we obtain the same upper bounds for BPP's. In the next section we will show that we essentially can not do better than that.

5 The lower bound

In this section we will show that model-checking BPP's w.r.t LTL is EXPSPACE-hard in the size of the system and PSPACE-hard in the size of the formula.

Theorem 5.1 *Model checking BPP w.r.t to LTL is PSPACE-hard in the size of the formula.*

This follows directly from the fact [17] that model-checking finite-state systems is PSPACE-complete in the size of the formula. We use the following problem to obtain a lower bound in the size of the system.

Problem 5.1 *Given a k-dimensional VASS $(\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ such that vectors in A have only components -1,0 and 1 and a control state $q \in Q$. Does there exist no transition sequence $(q_0, \vec{v}_0) \rightsquigarrow (q, \vec{v})$ for some $\vec{v}$?*

The following theorem follows from a result of Lipton [8], who essentially shows that there exists a Petri Net of dimension k such that vectors in A have only components -1,0 and 1 which can truthfully simulate a counter from 0 to 2^{2^k}. Thus for every Turing machine which uses exponential space we can construct a VASS such that q is not reachable iff the Turing machine accepts by simulating the Turing machine with a 3-counter machine as in [13].

Theorem 5.2 *Problem 5.1 is EXPSPACE-hard in the dimension of the VASS.*

Then, we are able to prove the following theorem where the size of a BPP is defined to be the number of dimensions.

Theorem 5.3 *Model-checking BPP w.r.t. LTL is EXPSPACE-hard in the size of the system.*

Proof: We show that Problem 5.1 is polynomial-time reducible to model-checking BPP w.r.t LTL. In order to do this we will construct from a given VASS a BPP $\mathcal{P}$ and a formula φ in LTL, such that $\mathcal{P}$ satisfies φ if and only if the control state q in the VASS can not be reached.

Let us fix a k-dimensional VASS $\mathcal{S} = (\Sigma, \vec{v}_0, A, Q, q_0, \delta)$ such that vectors in A have only components -1,0 and 1 and a control state q. The labelling of the transitions is not important for our construction. We can suppose that all the transitions are labelled differently. The transition vectors in $\mathcal{S}$ can contain more than one component which has value -1. Remember that in a BPP every transition contains at most one -1. Therefore we split every transition which contains k components of value -1 into k transitions, such that every transition vector contains only one -1 component and such that the overall effect remains the same. For example $q_1 \to (q_2, (-1,0,-1,-1,1,0))$ is split into $q_1 \to (q_3, (-1,0,0,0,1,0))$, $q_3 \to (q_4, (0,0,-1,0,0,0)$ and $q_4 \to (q_2, (0,0,0,-1,0,0))$. The intermediate states do not appear elsewhere in the constructed VASS. Furthermore we add a transition $q \xrightarrow{t} (q, \vec{0})$ to the VASS. Let us call the so constructed VASS $\mathcal{S}'$. The construction is polynomial in the size of $\mathcal{S}$. Now, $\mathcal{S}'$ can be obtained as a product of a BPP $\mathcal{P}$ and an automaton A. The automaton is just the control structure of the VASS, given by the transitions without vectors, whereas the BPP is the VASS, where all the control states are collapsed into one. Remember that all the transitions are labelled differently. Because of that, we can construct a LTL formula $\varphi = $ ("run of A" $\Rightarrow \Box\neg t$) which has as models all sequences of transition labels which if they are runs of the automaton A never contain t. So, finally we have that $\mathcal{P}$ satisfies φ iff there is no transition sequence in $\mathcal{S}$ such that $(q_0, \vec{v}_0) \rightsquigarrow (q, \vec{v})$ for some $\vec{v}$. $\Box$

A corollary of theorem 5.3 is that model-checking Petri Nets w.r.t. the linear time μ-calculus is also EXPSPACE-hard. Now we are able to give the main result.

5.1 Main result

Putting the results of chapter 4 and 5 together, we obtain the following theorem.

Theorem 5.4 *Model-checking BPP or Petri Nets w.r.t LTL or linear-time μ-calculus is EXPSPACE-complete in the size of the system and PSPACE-complete in the size of the formula.*

6 Conclusion

In this paper we have given a detailed complexity analysis of model checking Petri Nets or BPP's w.r.t linear-time μ-calculus or LTL. We have shown that model checking in all these cases is EXPSPACE-complete in the size of the system and PSPACE-complete in the size of the formula. Remark that for finite state systems model-checking is also PSPACE-complete in the size of the formula, whereas it is in linear time in the size of the system. It is somewhat surprising that there is no difference in the complexity for Petri Nets and BPP's, because BPP is a rather small subclass of Petri Nets. There has been some hope that model-checking would be easier for BPP's, but the intersection of a BPP and an automaton representing a formula is rather powerful. In fact, it corresponds to Parallel Pushdown automata (PPDA) [9] which form a class of infinite state systems whose expressiveness lies between BPP and Petri Nets. Our construction in chapter 5 essentially shows that reachability in PPDA is as hard as reachability in Petri Nets and therefore Petri Nets and BPP's have the same complexity for model checking w.r.t linear time logics.

Acknowledgement: The author would like to thank Javier Esparza for very helpful discussions on this topic.

References

1. P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127:91–101, 1996.
2. I. Borosh and L. Treybis. Bounds on positive integral solutions of linear Diophantine equations. *Proc. Amer. Math Soc.*, 55:299–304, 1976.
3. Søren Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, University of Edinburgh, 1993.
4. M. Dam. Fixed points of Büchi automata. In R. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *Lecture Notes in Computer Science*, pages 39–50. Springer-Verlag, 1992.
5. J. Esparza. On the decidability of model checking for several mu-calculi and petri nets. In *CAAP: Colloquium on Trees in Algebra and Programming*. LNCS 787, Springer-Verlag, 1994.
6. J. Esparza. More infinite results. In Steffen and Margaria, editors, *Infinity: International Workshop on Verification of Infinite State Systems*. Tech. Report MIP-9614, Univ. Passau, 1996.
7. J. Esparza and A. Kiehn. On the model checking problem for branching time logics and basic parallel processes. In *CAV'95*. LNCS 939, 1995.
8. R. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976.
9. Faron Moller. Infinite results. In *CONCUR: 7th International Conference on Concurrency Theory*, volume 1119. LNCS, Springer-Verlag, 1996.
10. D. Park. Concurrency and Automata on Infinite Sequences. In *5th GI-Conference on Theoretical Computer Science*. 1981. LNCS 104.
11. A. Pnueli. The Temporal Logic of Programs. In *FOCS'77*. IEEE, 1977.
12. Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, April 1978.
13. Louis E. Rosier and Hsu-Chun Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *Journal of Computer and System Sciences*, 32(1):105–135, February 1986.
14. S. Safra. On the complexity of ω-automata. In *29th Annual Symposium on Foundations of Computer Science*, pages 319–327, White Plains, New York, 24–26 October 1988. IEEE.
15. W. J. Savitch. Relational between nondeterministic and deterministic tape complexity. *Journal of Computer and System Sciences*, 4:177–192, 1970.
16. M. Y. Vardi. A temporal fixpoint calculus. In ACM-SIGPLAN ACM-SIGACT, editor, *Conference Record of the 15th Annual ACM Symposium on Principels of Programming Languages (POPL '88)*, pages 250–259, San Diego, CA, USA, January 1988. ACM Press.
17. M. Y. Vardi. An automata-theoretic approach to linear temporal logic (banff'94). *Lecture Notes in Computer Science*, 1043, 1996.
18. Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 15 November 1994.
19. P. Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56, 1983.

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