BRICS

Basic Research in Computer Science

BRICS RS-95-18 A. Cheng: Complexity Results for Model Checking

Complexity Results for Model Checking

Allan Cheng

BRICS Report Series **RS-95-18**

ISSN 0909-0878 **February 1995**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@daimi.aau.dk

BRICS publications are in general accessible through WWW and anonymous FTP:

`http://www.brics.aau.dk/BRICS/`
`ftp ftp.brics.aau.dk (cd pub/BRICS)`

Complexity Results for Model Checking*

Allan Cheng**

Computer Science Department
Cornell University
Ithaca, New York 14853, USA
e-mail:acheng@cs.cornell.edu

Abstract. The complexity of model checking branching and linear time temporal logics over Kripke structures has been addressed in e.g. [SC85, CES86]. In terms of the size of the Kripke model and the length of the formula, they show that the model checking problem is solvable in polynomial time for CTL and NP-complete for $\mathsf{L}(F)$. The model checking problem can be generalised by allowing more succinct descriptions of systems than Kripke structures. We investigate the complexity of the model checking problem when the instances of the problem consist of a formula and a description of a system whose state space is at most exponentially larger than the description. Based on Turing machines, we define *compact systems* as a general formalisation of such system descriptions. Examples of such compact systems are K-bounded Petri nets and synchronised automata, and in these cases the well-known algorithms presented in [SC85, CES86] would require exponential space in term of the sizes of the system descriptions and the formulas; we present polynomial space upper bounds for the model checking problem over compact systems and the logics CTL and $\mathsf{L}(X, U, S)$. As an example of an application of our general results we show that the model checking problems of both the branching time temporal logic CTL and the linear time temporal logics $\mathsf{L}(F)$ and $\mathsf{L}(X, U, S)$ over K-bounded Petri nets are PSPACE-complete.

1 Introduction

Formal verification techniques of distributed systems have received much attention, see for example [Lam80, SC85, CES86, Lar88, Mil89, SW89, Val90, WG93].

A predominant technique is known as *model checking*. The approach is as follows. The systems one considers either explicitly or implicitly specify a *state space* which can be regarded as a (labelled) graph. Viewing these graphs as *models* (Kripke structures) for *temporal logics*, one can use logical formulas to express properties of the graphs. The problem of verifying if a system satisfies

* This work has been supported by The Danish Research Councils and the Danish Research Academy.

** Visiting from Aarhus, **BRICS**, Basic Research in Computer Science, Centre of the Danish National Research Foundation. e-mail:acheng@daimi.aau.dk

a property encoded in a formula then reduces to the problem of checking if the formula is satisfied in the systems state space.

The complexity of the model checking problem for both *linear* and *branching time propositional temporal logics* has been investigated, among others, by Sistla and Clarke in [SC85] and by Clarke, Emerson, and Sistla in [CES86]. Both papers consider Kripke structures (or just structures) as models for the logics and the complexity results are stated in terms for the *sizes of the structures* and the length of the formulas. The paper [CES86] shows that the model checking problem for the computational tree logic CTL can be solved in polynomial time while [SC85] shows that the model checking problem for the linear time temporal logic $\mathsf{L}(F)$ is NP-complete.

There exists other well known classes of systems over which we can interpret such logics. K-bounded Petri nets and synchronised automata are examples of such systems. Common to these systems is that they can be viewed as *compact representations* of structures. More precisely, they can specify structures whose sizes are exponentially larger than the description of the systems. For example, the state space of a K-bounded Petri net can be (no more than) exponentially larger than the net. We will call such systems *compact systems*.

K-bounded Petri nets [JLL77] and synchronised automata [WG93] are examples of models which are widely use to specify and implement concurrent systems. Verification techniques for these and related systems have been presented in [Lar88, SW89, Val90, WG93, ES92, Esp93, BCM+92]. Whereas the work in [Lar88, SW89] focuses on algorithms (tableau systems) for solving the model checking problem, the work in [Val90, WG93, ES92, Esp93, BCM+92] is mainly motivated by the state space explosion problem and how to overcome this problem taking time, and especially, space consumption into account. Structures such as "stubborn sets", "persistent sets", "net unfoldings", and "Binary Decision Diagrams" are proposed.

It is therefore important to evaluate these verification techniques with respect to their space requirements. A primary measure for this evaluation is *space complexity*. For more specific problems connected to 1-safe Petri nets, for example for the reachability, liveness, and deadlock problem, such a study has been presented in [CEP93].

In this paper we choose a rather general setting both in terms of the systems considered and the problems to be solved. Namely, we formalise the notion of compact systems and investigate what happens to the complexity of the model checking problem in terms of the *size of a compact system* and the length of the formula. We show that for the well-known temporal logics CTL, $\mathsf{L}(F)$, and $\mathsf{L}(X, U, S)$ and any class of compact systems, the model checking problem is in PSPACE. Since for most nontrivial classes of compact systems the model checking problem for temporal logics containing the F (future) operator is usually PSPACE-hard, our results provide matching upper bounds. In contrast, the algorithms in [SC85, CES86] for CTL and $\mathsf{L}(F)$ would both require exponential space. As an example of how our results may be applied, we consider K-bounded Petri nets. From [CEP93] we easily conclude that the problems for CTL, $\mathsf{L}(F)$, and $\mathsf{L}(X, U, S)$ are PSPACE-hard. Our results then allow us to conclude that they that PSPACE-complete. In terms of the size of the state spaces of the K-bounded Petri nets (and the length of the formulas) the complexities are P [CES86], NP-complete and PSPACE-complete [SC85], respectively.

The paper is organised as follows. In Sect. 2 we give the necessary definitions. This includes compact systems, the logics and their interpretations, and the model checking problems. Then, in Sect. 3 we give the upper bounds on the model checking problems. In Sect. 4 we apply our results to K-bounded Petri nets and in Sect. 5 we conclude and give suggestions for future work.

2 Definitions

Intuitively, the systems we will consider all have the property that we can associate to them Kripke structures which corresponds to their state space. Moreover, the size of this Kripke structure is atmost exponential in the size of the description of the system. For example K-bounded Petri nets (Petri nets where each place can have atmost K tokens) have this property; a net with n places has atmost $(K+1)^n$ reachable states. We shall refer to these systems as *compact systems*. We continue by formalising compact systems, defining the logics $\mathsf{B}(X, U)$ (CTL) and $\mathsf{L}(X, U, S)$ and their interpretations, and finally the model checking problems.

2.1 The Compact Systems

A class of compact systems will be described using strings to encode the systems and a polynomial space bounded nondeterministic Turing machine which given an encoding of a system will "simulate" it. The state space of the Turing machine will then be used to define the state space of the input string (the compact system). In Sect. 4 we will present K-bounded Petri nets as a class of compact systems. For the ease of the presentation we assume that the reader is familiar with Turing machines and basic complexity theory, see [HU79] for an introduction.

Definition 1. A *class of compact systems*, $(\mathcal{C}, \mathcal{M}_\mathcal{C})$, is a set $\mathcal{C} = \{s_1, s_2, \ldots\}$ of strings referred to as systems, over some alphabet Σ, together with a polynomial space bounded nondeterministic Turing machine $\mathcal{M}_\mathcal{C}$ with a distinguished "signal" state $q^\star$ such that

- for any $s \in \mathcal{C}$, $\mathcal{M}_\mathcal{C}$ has a unique configuration[3] c_s, such that any computation on s reaches c_s and c_s is the first configuration along the computation whose machine state is $q^\star$. Intuitively, c_s is the initial state of the system s.
- for any string $s \notin \mathcal{C}$, $\mathcal{M}_\mathcal{C}$ will never enter the state $q^\star$ for any computation on input s.

[3] A configuration of $\mathcal{M}_\mathcal{C}$ consists of the contents of the tape, a machine state, and a postition on the tape.

In the following we assume a fixed class of compact systems $(\mathcal{C}, \mathcal{M}_\mathcal{C})$. Since $\mathcal{M}_\mathcal{C}$ is polynomial space bounded on inputs of length n, say by the polynomial $q'(n)$, $\mathcal{M}_\mathcal{C}$ has on any input s atmost exponentially many reachable configurations whose machine state is $q^\star$; there exists a $B > 0$ and a polynomial $q(n)$, independent of s, such that given s there are atmost $B^{q(|s|)}$ possible configurations. Call the configurations whose machine state is $q^\star$ *signal configurations*, and let $Sig_{\mathcal{M}_\mathcal{C}}(s) = \{c \,|\, c \text{ is a signal configuration of } \mathcal{M}_\mathcal{C} \text{ on input } s\}$. We can now define the state space associated to $s \in \mathcal{C}$.

Definition 2. For $s \in \mathcal{C}$, let (V_s, E_s, i_s) be the rooted graph whose nodes V_s are $Sig_{\mathcal{M}_\mathcal{C}}(s)$, the signal configurations of $\mathcal{M}_\mathcal{C}$ on input s, whose edges are pairs of nodes (c, c') such that $(c, c') \in E_s$ if and only if $\mathcal{M}_\mathcal{C}$ can reach c' from c without entering any other signal configurations, and whose initial/root node is the unique configuration c_s.

Remark. We shall refer to (V_s, E_s, i_s) as the *state space of the system s*, whenever $s \in \mathcal{C}$. Also, the nodes will be refered to as states of s and will be ranged over by $v, w, \ldots$. Notice that any system s has atmost $B^{q(|s|)}$ states. Whenever (V_s, E_s, i_s) is understood from the context, we will use the notation $v_0 \longrightarrow v_1 \longrightarrow \cdots \longrightarrow v_n$ instead of $(v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n) \in E_s$. For a state v of s, we use the notation $v \not\longrightarrow$ to indicate that there exists no state $v' \in V_s$ such that $v \longrightarrow v'$.

A run of the system s is any sequence $v_0 \longrightarrow v_1 \longrightarrow \cdots$ that is either infinite or ends in a state v_n such that $v_n \not\longrightarrow$. The length of a finite run $v_0 \longrightarrow \cdots \longrightarrow v_n$ is n. We will use Greek letters $\sigma, \gamma, \ldots$ to denote runs of the system s.

Henceforth, we assume a fixed class of compact systems $(\mathcal{C}, \mathcal{M}_\mathcal{C})$ and continue by giving the syntax of the temporal logics we will be considering.

2.2 The Logics

Let $\mathcal{A}$ be a set of *atomic propositions*. We assume it to be fixed in the following. We will consider the temporal logics $\mathsf{L}(X, U, S)$ and $\mathsf{B}(X, U)$, both described in detail in [Eme90].

The formulas of the logic $\mathsf{L}(X, U, S)$ over $\mathcal{A}$ are defined inductively:

- t, f, or any $p \in \mathcal{A}$.
- $\neg\phi_1$, $\phi_1 \wedge \phi_2$, $X(\phi_1)$, $\phi_1 U \phi_2$, and $\phi_1 S \phi_2$ are formulas, where ϕ_1 and ϕ_2 are formulas.

$\mathsf{L}(X, U, S)$ is a linear time temporal logic, whose formulas are interpreted over runs of a systems s.

In order to interpret the formulas of $\mathsf{L}(X, U, S)$ over the runs of a system s we need a valuation $\eta_s : \mathcal{A} \times Sig_{\mathcal{M}_\mathcal{C}}(s) \longrightarrow \mathbf{2}$ which tells us if an atomic proposition holds at a state. Furthermore, we require that the valuation eta_s is computable by a polynomial space bounded deterministic Turing machine which we denote T_{η_s}. We assume the atomic propositions to be encoded as strings.

Given s, $\sigma = v_0 \longrightarrow v_1 \longrightarrow \cdots$ a run of s, a natural number $0 \leq i \leq |\sigma|$, and η_s. Relative to η_s we interpret the formulas of $\mathsf{L}(X, U, S)$ at (σ, i) as follows:

- $(\sigma, i) \models t$ and $(\sigma, i) \not\models f$.
- $(\sigma, i) \models p$ iff $\eta_s(p, v_i) = \mathbf{1}$.
- $(\sigma, i) \models \neg\phi$ iff $(\sigma, i) \not\models \phi$.
- $(\sigma, i) \models \phi_1 \wedge \phi_2$ iff $(\sigma, i) \models \phi_1$ and $(\sigma, i) \models \phi_2$.
- $(\sigma, i) \models X(\phi)$ iff $i < |sigma|$ and $(\sigma, i+1) \models \phi$.
- $(\sigma, i) \models \phi_1 U \phi_2$ iff there exists a natural number $i \leq j \leq |\sigma|$ such that $(\sigma, j) \models \phi_2$ and for all $i \leq k < j$, $(\sigma, k) \models \phi_1$.
- $(\sigma, i) \models \phi_1 S \phi_2$ iff there exists a $0 \leq j \leq i$ such that $(\sigma, j) \models \phi_2$ and for all $j < k \leq i$, $(\sigma, k) \models \phi_1$.

The interpretation of the logic should be clear except perhaps for $\phi_1 U \phi_2$ and $\phi_1 S \phi_2$. Intuitively the former expresses that ϕ_2 holds somewhere in the future and that ϕ_1 holds "until" then; the latter expresses that somewhere in the past ϕ_2 holds and "since" then, ϕ_1 holds. Remember that the "past" and "future" is relative to σ.

Next, we consider a well-known branching time temporal logic. The formulas of the logic $\mathsf{B}(X, U)$ over $\mathcal{A}$ (also known as CTL [CES86]) are also defined inductively:

- t, f, or any $p \in \mathcal{A}$.
- $\neg\phi_1$, $\phi_1 \wedge \phi_2$, $EX(\phi_1)$, $AX(\phi_1)$, $E(\phi_1 U \phi_2)$, or $A(\phi_1 S \phi_2)$ are formulas, where ϕ_1 and ϕ_2 are formulas.

$\mathsf{B}(X, U)$ is a branching time logic whose formulas are interpreted at state of a system s.

Given a system s, v a state of s, and a valuation η_s. Then, the formulas of $\mathsf{B}(X, U)$ are interpreted relative to η_a as follows:

- $v \models t$ and $v \not\models f$.
- $v \models p$ iff $\eta_s(p, v) = \mathbf{1}$.
- $v \models \neg\phi$ iff $v \not\models \phi$.
- $v \models \phi_1 \wedge \phi_2$ iff $v \models \phi_1$ and $v \models \phi_2$.
- $v \models EX(\phi)$ iff there exists $v \rightarrow v'$ such that $v' \models \phi$.
- $v \models AX(\phi)$ iff for all $v \rightarrow v'$, $v' \models \phi$.
- $v \models E(\phi_1 U \phi_2)$ iff there exists $v \rightarrow v_1 \rightarrow \cdots \rightarrow v_n$ such that $v_n \models \phi_2$ and for all $0 \leq j < n$, $v_j \models \phi_1$, where $v_0 = v$.
- $v \models A(\phi_1 U \phi_2)$ iff for all $v \rightarrow v_1 \rightarrow \cdots$ there exists an n such that $v_n \models \phi_2$ and for all $0 \leq j < n$, $v_j \models \phi_1$, where $v_0 = v$.

The interpretation of the temporal formulas shows the branching nature of $\mathsf{B}(X, U)$. At a state several possible successor states or paths have to be taken into account. The intuition behind the interpretation of the "until" formulas corresponds well to that of $\mathsf{L}(X, U, S)$ except that we quantify existentially or universally over paths from the state v.

2.3 The Model Checking Problems

Definition 3. An instance of the *model checking problem for* $\mathsf{L}(X, U, S)$ is a tuple (s, T_{η_s}, ϕ), where $s \in \mathcal{C}$, η_s is a valuation, and ϕ is a $\mathsf{L}(X, U, S)$ formula. The model checking problem for (s, T_{η_s}, ϕ) is to decide whether or not there exists a run σ of the system s such that $(\sigma, 0) \models \phi$.

Definition 4. An instance of the *model checking problem for* $\mathsf{B}(X, U)$ is a tuple (s, T_{η_s}, ϕ), where $s \in \mathcal{C}$, η_s is a function, and ϕ is a $\mathsf{B}(X, U)$ formula. The model checking problem for (s, T_{η_s}, ϕ) is to decide whether or not $i_s \models \phi$.

Remark. Notice that the valuation is usually implicitely assumed part of the problem instances to the model checking problem [SC85] or assume to be easily computable [CES86]. We could also have defined a valuation to be relative to $(\mathcal{C}, \mathcal{M}_{\mathcal{C}})$ and $\mathcal{A}$. This wouldn't leed to any significant changes of the results.

Having given the necessary definitions, we summarize in the table below our results and related known results [SC85, CES86] about the model checking problem.

Logic	**Problem Instance**	**Complexity**
CTL	R-structure (Kripke) and a formula	P
$\mathsf{L}(F)$	R-structure and a formula	NP-complete
$\mathsf{L}(X, U, S)$	R-structure and a formula	PSPACE-complete
CTL $\mathsf{L}(F)$ $\mathsf{L}(X, U, S)$	Compact system and a formula	PSPACE

Fig. 1. Complexity in terms of Kripke structures and compact systems.

3 The Upper Bounds

In this section we provide PSPACE upper bounds for the model checking problems defined in Def. 3 and Def. 4. We start by $\mathsf{L}(X, U, S)$.

3.1 Linear Time

The results of this section is based on the idea behind the decision procedure for the logic $\mathsf{L}(X, U, S)$ given in [SC85]. There, Sistla and Clarke also reduce the problem of determining truth in an R-structure (model checking problem over Kripke structures) to the satisfiability problem by encoding an R-structure into a formula. Since a compact system may have exponentially many states, encoding the state space (Kripke structure) of a system in a formula would yield an exponentially long formula.

Instead one could try to encode the system itself in the logic. This is easily done for systems like 1-safe Petri nets. However, when considering other models like K-bounded Petri nets, this encoding quickly becomes more troublesome. One of the reasons why we have chosen the setting of $(\mathcal{C}, \mathcal{M}_{\mathcal{C}})$ is because describing a class of systems as a class of compact sytems is often more straightforward and credible.

Our solution is based on proving the existence of a run which ends in a dead state v', $v' \not\rightarrow$, or in a loop. Moreover, one has to bound the lengths of these paths. This seems to be the only (obvious) technique applicable to solve the problem.

Definition 5. Given a system s, $\sigma = v_0 \rightarrow v_1 \rightarrow \cdots$ a run of s, ϕ a formula of $\mathsf{L}(X, U, S)$, and η_s a valuation. Then, $Sub(\sigma, i, \phi)$ is the set of subformulas ϕ' of ϕ such that $(\sigma, i) \models \phi'$.

Lemma 6. *Given a system s, $\sigma = v_0 \rightarrow v_1 \rightarrow \cdots$ a run of s, ϕ a formula of $\mathsf{L}(X, U, S)$, and η_s. If $Sub(\sigma, i, \phi) = Sub(\sigma, j, \phi)$, $i < j \leq |\sigma|$, and $v_i = v_j$, then*

$$\forall 0 \leq l \leq i.\, Sub(\sigma, l, \phi) = Sub(\sigma', l, \phi)$$

$$\forall j \leq l \leq |\sigma|.\, Sub(\sigma, l, \phi) = Sub(\sigma', l - (j - i), \phi)$$

where all indices range over natural numbers and $\sigma' = v'_0 \rightarrow v'_1 \rightarrow \cdots = v_0 \rightarrow \cdots \rightarrow v_i \rightarrow v_{j+1} \rightarrow v_{j+2} \rightarrow \cdots$.

Proof. Induction in ϕ. □

Given a run $\sigma = v_0 \rightarrow v_1 \rightarrow \cdots$ of a system s, two indices i and j, and a formula ϕ of $\mathsf{L}(X, U, S)$. ϕ is said to be fulfilled between i and j if and only if $i < j$ and there exists a k, $i \leq k < j$, such that $(\sigma, k) \models \phi$.

Lemma 7. *Given a systems s, $\sigma = v_0 \rightarrow v_1 \rightarrow \cdots$ an infinite run of s, ϕ a formula of $\mathsf{L}(X, U, S)$, an index i, and a natural number $p > 0$ (the period) such that $v_i = v_{i+p}$, $Sub(\sigma, i, \phi) = Sub(\sigma, i + p, \phi)$, and for every formula $\phi_1 U \phi_2$ in $Sub(\sigma, i, \phi)$, ϕ_2 is fulfilled between i and $i+p$. Let $\sigma' = v'_0 \rightarrow v'_1 \rightarrow \cdots$ be the run $v_0 \rightarrow \ldots \rightarrow v_i \rightarrow \cdots \rightarrow v_{i+p-1} \rightarrow v_i \rightarrow \cdots \rightarrow v_{i+p-1} \rightarrow v_i \rightarrow \cdots$, i.e. the period $v_i \rightarrow \cdots \rightarrow v_{i+p}$ is repeated infinitely often. Then*

$$\forall 0 \leq l \leq i + p.\, Sub(\sigma, l, \phi) = Sub(\sigma', l, \phi)$$

$$\forall i \leq l.\, Sub(\sigma', l, \phi) = Sub(\sigma', (l + p), \phi)$$

Proof. Induction in ϕ, case based analysis. □

Lemma 7 states that from a run where two identical states satisfy the same subformulas of ϕ, one can obtain a new run which consists of a finite "prefix" and a period which is repeated.

Now we continue to the theorem which our upper bound result is based upon. It is a variant of "Ultimately Periodic Model Theorem" [SC85].

Theorem 8. *Given a system s, η_s a valuation, σ a run of s, and ϕ a formula of $\mathsf{L}(X,U,S)$ such that $(\sigma,0)\models\phi$, then*

1) if $|\sigma|<\infty$, then there exists a run γ of s such that $|\gamma|\le B^{q(|s|)}2^{|\phi|}$ and $(\gamma,0)\models\phi$.
2) if $|\sigma|=\infty$, then there exists a run $\gamma=w_0\to w_1\to\cdots$ and indices $0\le i\le B^{q(|s|)}2^{|\phi|}$, $0<p\le|\phi|B^{q(|s|)}2^{|\phi|}$ such that $w_l=w_{l+p}$ for $l\ge i$ and $(\gamma,0)\models\phi$.

where B and $q(n)$ were described in Sect. 2.1.

Proof. We only consider the case where σ is infinite. The other case can be handled similarly. So assume $(\sigma,0)\models\phi$. Since s has at most $B^{q(|s|)}$ states and ϕ has at most $2^{|\phi|}$ subformulas there must exist indices $i<j$ such that

i) $v_i=v_j$, $Sub(\sigma,i,\phi)=Sub(\sigma,j,\phi)$, and for all $\phi_1U\phi_2\in Sub(\sigma,i,\phi)$, ϕ_2 is fulfilled between i and j.

Now applying Lemma 6 repeatedly to the initial part of σ we obtain a new run σ' and indices $i'<j'$ such that

ii) $(\sigma',0)\models\phi$, $v'_{i'}=v'_{j'}$, $Sub(\sigma,i,\phi)=Sub(\sigma',i',\phi)=Sub(\sigma',j',\phi)$, $i'\le B^{q(|s|)}2^{|\phi|}$, and for all $\phi_1U\phi_2\in Sub(\sigma',i',\phi)$, ϕ_2 is fulfilled between i' and j'.

Also, repeatedly applying Lemma 6 between $v'_{i'}$ and $v'_{j'}$ we can assume that $(j'-i')\le|\phi|B^{q(|s|)}2^{|\phi|}$; if at any time during the application of Lemma 6 the current value of $(j'-i')$ is larger than $|\phi|B^{q(|s|)}2^{|\phi|}$, there must exist more than $|\phi|$ identical states v'_l among $v'_{i'+1},\ldots,v'_{j'}$ satisfying the same set $Sub(\sigma',l,\phi)$ of formulas. Since ϕ has less then $|\phi|$ subformulas of the form $\phi_1U\phi_2$, we can remove a loop between two of these states and still have all such ϕ_2's fulfilled between i' and (the new value of) j' (in the resulting new run). We therefore conclude that there exists a run σ'' of s and indices $i''<j''$ such that

iii) $(\sigma'',0)\models\phi$, $v''_{i''}=v''_{j''}$, $Sub(\sigma'',i'',\phi)=Sub(\sigma'',j'',\phi)$, $i''\le B^{q(|s|)}2^{|\phi|}$, $(j''-i'')\le|\phi|B^{q(|s|)}2^{|\phi|}$, and for all $\phi_1U\phi_2\in Sub(\sigma'',i'',\phi)$, ϕ_2 is fulfilled between i'' and j''.

Using Lemma 7 we conclude that the run $\gamma=v''_0\to\cdots\to v''_{i''}\to\cdots\to v''_{j''}\to v''_{i''+1}\to\cdots\to v''_{j''}\to\cdots$ of s has the property $(\gamma,0)\models\phi$. □

Theorem 9. *Fix any class of compact systems $(\mathcal{C},\mathcal{M}_\mathcal{C})$ and set of atomic propositions $\mathcal{A}$. Then, the model checking problem for $\mathsf{L}(X,U,S)$ is in PSPACE.*

Proof. We shall describe an algorithm in a Pascal like programming language which given any instance (s, T_{η_s}, ϕ) of the model checking problem for L(X,U,S) solves it using only an amount of space polynomial in the sum $|s| + |T_{\eta_s}| + |\phi|$. Let n denote this sum. Henceforth, whenever we write polynomial, we implicitly mean polynomial in n. The algorithm can be encoded as a nondeterministic polynomial space bounded Turing machine.

Notice that given possible configurations v and v' of $\mathcal{M}_\mathcal{C}$ run on input s, there are polynomial space bounded nondeterministic Turing machines which decide properties such as whether or not $v \in V_s$, $v \rightarrow v'$ given $v \in V_s$, or $v \not\rightarrow$. Moreover, we can also compute i_s given s using only a polynomial amount of space. For example, let us consider the case where we given $v \in V_s$ and a possible configuration v' have to decide if $v \rightarrow v'$. Our nondeterministic Turing machine will simulate $\mathcal{M}_\mathcal{C}$ from v. It guesses a number $k_1 \leq B^{q(|s|)}$. v' is reachable from v if and only if it can be reached from v in at most $B^{q(|s|)}$ computation steps of $\mathcal{M}_\mathcal{C}$ (simulated on input s). Then, storing at most two new configurations of $\mathcal{M}_\mathcal{C}$, it guesses, one step at the time, a computation of $\mathcal{M}_\mathcal{C}$ of length k_1. For each step it decrements k_1 and checks if $\mathcal{M}_\mathcal{C}$ can go from the old configuration to the new (guessed) configuration. Also, it checks that none of these intermediate configurations of $\mathcal{M}_\mathcal{C}$ are signal configurations. Finally, if it reaches $k_1 = 0$ it check that the guessed configuration equal v' and is a signal configuration. It should be clear that this machine uses at most a polynomial amount of space. Now applying the technique from [HU79] Theorem 12.10, we obtain a deterministic polynomial space bounded Turing machine, since we can compute an exponential bound for the maximal number of configurations of the nondeterministic machine. Also, since $\eta_s(p, v)$ can be computed in polynomial space by T_{η_s} this can also be done by simulating T_{η_s}.

Hence, in the algorithm we shall refer freely to the states of s and use the notation $v \rightarrow v'$. Whenever a property is checked and is found not to hold, the algorithms fails. Let $Sub(\phi)$ be the set of subformulas of ϕ, let $\mathcal{S}$ range over subsets of $Sub(\phi)$, and let $Un(\phi)$ be the set $\{\phi_2 \mid \phi_1 U \phi_2 \in Sub(\phi)\}$.

First we consider the case where the answer to (s, T_{η_s}, ϕ) is YES because of an infinite run.

<u>ALGORITHM 1A</u>

Guess $0 \leq n_1 \leq B^{q(|s|)} 2^{|\phi|}$
Guess $\mathcal{S}_{\text{current}} \subseteq Sub(\phi)$
Let $v_{\text{current}} = i_s$
Check boolean consistency of $\mathcal{S}_{\text{current}}$ **and** v_{current}**, i.e. that**
    $(\forall p \in Sub(\phi).\, s \in \mathcal{S}_{\text{current}} \Leftrightarrow \eta_s(p, v_{\text{current}}))$
    $(\forall \phi_1 \wedge \phi_2 \in Sub(\phi).\, \phi_1 \wedge \phi_2 \in \mathcal{S}_{\text{current}} \Leftrightarrow \phi_1 \in \mathcal{S}_{\text{current}}$ **and** $\phi_2 \in \mathcal{S}_{\text{current}})$
    $(\forall \neg\phi' \in Sub(\phi).\, \neg\phi' \in \mathcal{S}_{\text{current}} \Leftrightarrow \phi' \notin \mathcal{S}_{\text{current}})$
Check that $\phi \in \mathcal{S}_{\text{current}}$
Check that $(\forall \phi_1 S \phi_2 \in Sub(\phi).\, \phi_2 \in \mathcal{S}_{\text{current}})$
Let $count := 0$
While $count < n_1$ **do**

12:   **Guess a configuration** v_{next} **and check that** $v_{\text{current}} \to v_{\text{next}}$
13:   **Guess** $\mathcal{S}_{\text{next}} \subseteq Sub(\phi)$
14:   **Check boolean consistency of** $\mathcal{S}_{\text{next}}$ **and** v_{next}
15:   **Check that**
16: $(\forall X\phi' \in Sub(\phi).\, X\phi' \in \mathcal{S}_{\text{current}} \Leftrightarrow \phi' \in \mathcal{S}_{\text{next}})$
17: $(\forall \phi_1 S \phi_2 \in Sub(\phi).\, \phi_1 S \phi_2 \in \mathcal{S}_{\text{next}} \Leftrightarrow$
            $(\phi_2 \in \mathcal{S}_{\text{next}} \vee (\phi_1 \in \mathcal{S}_{\text{next}} \wedge \phi_1 S \phi_2 \in \mathcal{S}_{\text{current}})))$
18: $(\forall \phi_1 U \phi_2 \in Sub(\phi).\, \phi_1 U \phi_2 \in \mathcal{S}_{\text{current}} \Leftrightarrow$
            $(\phi_2 \in \mathcal{S}_{\text{current}} \vee (\phi_1 \in \mathcal{S}_{\text{current}} \wedge \phi_1 U \phi_2 \in \mathcal{S}_{\text{next}})))$
19:   **Let** $count := count + 1$
20:   **Let** $v_{\text{current}} = v_{\text{next}}$
21:   **Let** $\mathcal{S}_{\text{current}} := \mathcal{S}_{\text{next}}$
22: Endwhile
23: Let $v_{\text{loop}} := v_{\text{current}}$
24: Let $\mathcal{S}_{\text{loop}} := \mathcal{S}_{\text{current}}$
25: Guess $0 \leq n_1 \leq |\phi| B^{q(|s|)} 2^{|\phi|}$
26: Let $count := 0$
27: Let $\mathcal{S}_U := \emptyset$
28: While $count < n_1$ **do**
29: (* Lines 29 to 38 are a copy of lines 12 to 21 *)
39:   **Let** $\mathcal{S}_U := \mathcal{S}_U \cup (\mathcal{S}_{\text{current}} \cap Un(\phi))$
40: Endwhile
41: Check that $v_{\text{current}} = v_{\text{loop}}$
42: Check that $\mathcal{S}_{\text{current}} = \mathcal{S}_{\text{loop}}$
43: Check that $(\forall \phi_1 U \phi_2 \in \mathcal{S}_{\text{loop}}.\, \phi_2 \in \mathcal{S}_U)$
44: Answer YES

For the case where the run is finite the following algorithm is derived:

<u>ALGORITHM 1B</u>

01: (* Lines 02 to 23 are a copy of lines 01 to 22 of ALGORITHM 1A ***)**
24: Check that $v_{\text{current}} \not\to$
25: Check that $(\forall \phi_1 U \phi_2 \in \mathcal{S}_{\text{current}}.\, \phi_2 \in \mathcal{S}_{\text{current}})$
26: Answer YES

Our final algorithm chooses nondeterministically between ALGORITHM 1A and ALGORITHM 1B. The correctness of the algorithm is straightforward to establish; if the algorithm answers YES, examine the guessed run of s, the values of $\mathcal{S}_{\text{current}}$ and $\mathcal{S}_U$, and conclude that this run indeed shows that the answer to (s, T_{η_s}, ϕ) is YES. Let σ' denote the guessed run. Then, by induction in $\phi' \in Sub(\phi)$ show that for any of the values $v_{\text{current}}(= v'_j)$ and $\mathcal{S}_{\text{current}}$, $(\sigma', j) \models \phi' \Leftrightarrow \phi' \in \mathcal{S}_{\text{current}}$; conversely, if the answer to (s, T_{η_s}, ϕ) is YES then by Theorem 8 there exists a run of the algorithm which answers YES.

I should be clear, from the assumptions about $\mathcal{M}_{\mathcal{C}}$, T_{η_s}, and the above comments about how to decide properties about the nodes and edges of (V_s, E_s)

using only a polynomial amount of space, that this algorithm can be implemented by a nondeterministic polynomial space bounded Turing machine. By Savitch's Theorem (NPSPACE = PSPACE) we conclude that the model checking problem for $\mathsf{L}(X, U, S)$ is in PSPACE. $\square$

3.2 Branching Time

In this section we describe an algorithm which will solve the model checking problem for $\mathsf{B}(X, U)$ using only a polynomial amount of space.

We start by sketching the intuition behind our solution. We will describe our solution stepwise using Turing machines and results about the complexity of constructing and transforming such machines.

The idea behind our construction will be to construct a C-tape Turing machine $\mathcal{M}$, where C is some constant. Denote these tapes by $T_1, \ldots, T_C$. Given a problem instance as input $\mathcal{M}$ will, in polynomial time, construct a (description of a) deterministic polynomial space bounded machine $M_{\phi'}$ for each occurrence of a subformula[4] ϕ' of ϕ. Let us call these machines *subformula machines*.

The subformula machines have the extra ability that they may call certain subformula machines as subroutines. A machine $M_{\phi'}$ will call as subroutines the subformula machines corresponding to the immediate subformulas of ϕ'; e.g. $M_{\phi_1 U \phi_2}$ will call M_{ϕ_1} and M_{ϕ_2} as subroutines. These calls may be considered as single computation steps from the calling machines point of view much in the same way as Turing machines use oracles. $\mathcal{M}$ will simulate all of the subformula machines and their subroutine calls. Notice that $\mathcal{M}$ will need a stack of depth at most $|\phi|$ to simulate the subroutine call sequence. Figure 2 illustrates this.

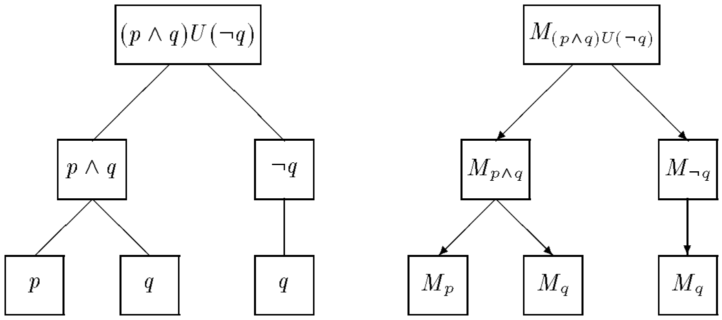

Fig. 2. Formula $(p \wedge q)U(\neg q)$ and the corresponding subformula machines. The arrows indicate which machines may be used as subroutines.

The subformula machines will be constructed in an bottom up fashion, i.e. when constructing $M_{\phi'}$, all subformula machines corresponding to the proper

[4] For convenience we shall use the same notation for both subformulas of and occurrences of subformulas of ϕ.

subformulas of ϕ' have been constructed. The subformula machines will be stored on one of $\mathcal{M}$'s tape using only a polynomial amount of tape. Each of these machines will compute the function $f_{\phi'} : V_s \longrightarrow \mathbf{2}$, defined by $f_{\phi'}(v) = \mathbf{1}$ if and only if $v \models \phi'$. Since there are at most $|\phi|$ subformula occurrences in ϕ, say m, one of $\mathcal{M}$'s C tapes, say T_l, will be used to simulate tapes for each of the machines $M_{\phi'}$; assuming we have the m machines enumerated, the j'th machines tape will consist of all of T_l's cells whose index i equals j modulo m. Figure 3 illustrates this for the case where there are three subformula occurrences. The j'th cell on a simulated tape is indicated by superscript j.

T_l:	$T^1_{\phi_1}$	$T^1_{\phi_2}$	$T^1_{\phi_3}$	$T^2_{\phi_1}$	$T^2_{\phi_2}$	$T^2_{\phi_3}$	$T^3_{\phi_1}$	$T^3_{\phi_2}$	-----

Fig. 3. T_l simulates 3 tapes: T_{ϕ_1}, T_{ϕ_2}, and T_{ϕ_3}.

Having constructed all the machines $M_{\phi'}$, $\mathcal{M}$ then proceeds by giving M_ϕ the input i_s and starts simulating M_ϕ.

Theorem 10. *Fix any class of compact systems $(\mathcal{C}, \mathcal{M}_\mathcal{C})$ and set of atomic propositions $\mathcal{A}$. Then, the model checking problem for $\mathsf{B}(X, U)$ is in PSPACE.*

Proof. Given (s, T_{η_s}, ϕ). Let n denote the size of this problem instance. $\mathcal{M}$ is defined as follows:

- $\mathcal{M}$ stores its input (s, η_s, ϕ) on tape T_1.
- It enumerates all subformulas occurrences of ϕ and remembers how many there are. This is done on T_2.

  Remark. Having found the number of subformula occurrences $\mathcal{M}$ will use T_3 to simulate one tape $T_{\phi'}$ for each subformula occurrence ϕ' as indicated in Fig.3.

- It now proceeds to construct the deterministic subformula machines as follows, starting with the smallest subformula occurrences. For a subformula occurrence ϕ':
  - If ϕ' is an atomic proposition p, then $M_{\phi'}$ is the machine which given v computes uses T_{η_s} to compute $\eta_s(p, v)$.
  - If ϕ' is of the form $\neg\phi''$, then $M_{\phi'}$ is the machine which given v calls $M_{\phi''}$ as subroutine with v as parameter and returns the negated value obtained from $M_{\phi''}$.
  - If ϕ' is of the form $\phi_1 \wedge \phi_2$, then $M_{\phi'}$ is the machine which given v first calls M_{ϕ_1} and then M_{ϕ_2}, remembering the values returned. It then returns $\mathbf{1}$ if and only if both values were $\mathbf{1}$.

- If ϕ' is of the form $EX(\phi'')$, then $\mathcal{M}$ first constructs a nondeterministic machine which does the following:

  Given v it guesses v', a possible configuration of s. Then it check that $v \rightarrow v'$. Finally it calls $M_{\phi''}$ with parameter v'. If the returned value is **1** it returns **1**, else it returns **0**.

  Then, $\mathcal{M}$ constructs a deterministic version of this machine using e.g. a construction similar to the one sketched in [HU79] Theorem 12.10. This can be done since a (fully space constructible) polynomial bound on the space consumption of the nondeterministic machine can be computed. This machine will systematically examine all successor states of v and check if ϕ'' holds at one of these states; if yes, then **1** is returned; if no, **0** is returned. This is the machine $M_{\phi'}$.
- If ϕ' is of the form $AX(\phi'')$, then $\mathcal{M}$ first constructs a nondeterministic machine which does the following:

  Given v it guesses v', a possible configuration of s. Then it check that $v \rightarrow v'$. Finally it calls $M_{\phi''}$ with parameter v'. If the returned value is **0** it returns **0**, else it returns **1**.

  Then, $\mathcal{M}$ constructs a deterministic version of this machine. This is the machine $M_{\phi'}$.
- If ϕ' is of the form $E(\phi_1 U \phi_2)$, then $\mathcal{M}$ first constructs a nondeterministic machine which does the following, given v:

  Guess a number $0 \leq k \leq B^{q(|s|)}$
  Let $v_{\text{current}} := v$
  While $k > 0$ **do**
      Call M_{ϕ_1} **to check that** $v_{\text{current}} \models \phi_1$
      Guess v'**, a possible state of** s
      Check that $v_{\text{current}} \rightarrow v'$
      Let $v_{\text{current}} := v'$
      Decrement k **by 1**
  Endwhile
  Call M_{ϕ_2} **to check that** $v_{\text{current}} \models \phi_2$

  If the machine reaches the last test and it is successful, the machine returns **1**, else it returns **0**.

  Then, $\mathcal{M}$ constructs a deterministic version of this machine. This is the machine $M_{\phi'}$.
- If ϕ' is of the form $A(\phi_1 U \phi_2)$, then $\mathcal{M}$ first constructs a nondeterministic machine which does the following, given v:

  It nondeterministically chooses to execute one of the following (nondeterministic) programs:

a) **Guess a number** $0 \leq k \leq B^{q(|s|)}$
Let $v_{\text{current}} := v$
While $k > 0$ **do**
  Call M_{ϕ_1} **to check that** $v_{\text{current}} \models \phi_1$
  Call M_{ϕ_2} **to check that** $v_{\text{current}} \not\models \phi_2$
  Guess v'**, a possible state of** s
  Check that $v_{\text{current}} \rightarrow v'$
  Let $v_{\text{current}} := v'$
  Decrement k **by 1**
Endwhile
Check that either
  $v_{\text{current}} \not\rightarrow$ **and**
  $v_{\text{current}} \models \phi_1$ **(by calling** M_{ϕ_1}**)and**
  $v_{\text{current}} \not\models \phi_2$ **(by calling** M_{ϕ_2}**)**
or
  $v_{\text{current}} \not\models \phi_1$ **and**
  $v_{\text{current}} \not\models \phi_2$

If the machine reaches the last test and it was successful, the machine returns **0**, else it returns **1**.

b) **Guess a number** $0 \leq k \leq 2B^{q(|s|)}$
Guess a possible configuration v_{loop}
Let $b := \mathbf{0}$
Let $v_{\text{current}} := v$
While $k > 0$ **do**
  If $v_{\text{current}} = v_{\text{loop}}$ **then let** $b := \mathbf{1}$
  Call M_{ϕ_1} **to check that** $v_{\text{current}} \models \phi_1$
  Call M_{ϕ_2} **to check that** $v_{\text{current}} \not\models \phi_2$
  Guess v'**, a possible state of** s
  Check that $v_{\text{current}} \rightarrow v'$
  Let $v_{\text{current}} := v'$
  Decrement k **by 1**
Endwhile
Check that $v_{\text{current}} = v_{\text{loop}}$ **and that** $b = \mathbf{1}$

If the machine reaches the last test and it was successful, the machine returns **0**, else it returns **1**.

Then, $\mathcal{M}$ constructs a deterministic version of this machine. This is the machine $M_{\phi'}$.

Remark. Notice that all nondeterministic machines above always answer either **1** or **0**, as do their deterministic versions. The algorithm for constructing the subformula machines is based on the observation that 1) if

$v \models E(\phi_1 U \phi_2)$, then there exists a path $v_0 \to v_1 \to \cdots \to v_k$ starting at v such that $(\forall 0 \leq i < k. v_i \models \phi_1)$, $v_k \models \phi_2$, and $0 \leq k \leq B^{q(|s|)}$; and 2) if $v \not\models A(\phi_1 A \phi_2)$, then 2a) there exists a path $v_0 \to v_1 \to \cdots \to v_k$ starting at v such that $(\forall 0 \leq i < k. v_i \models \phi_1 \wedge \neg\phi_2)$, $0 \leq k \leq B^{q(|s|)}$, and either $(v_k \models \neg\phi_1 \wedge \neg\phi_2)$ or $(v_k \not\to \wedge\, v_k \models \phi_1 \wedge \neg\phi_2)$; or 2b) there exists paths of the form $v_0 \to \cdots \to v_{k_1}$ and $v'_0 \to \cdots \to v'_{k_2}$ such that $v_0 = v$, $v_{k_1} = v'_0 = v'_{k_2}$, $0 \leq k_1, k_2 \leq B^{q(|s|)}$, and $(\forall 0 \leq j \leq k_1. v_j \models \phi_1 \wedge \neg\phi_2) \wedge (\forall 0 \leq j \leq k_2. v'_j \models \phi_1 \wedge \neg\phi_2)$.
By an induction argument one can show that the machine $M_{\phi'}$ indeed computes $f_{\phi'}$.

- Having constructed all subformula machines, $\mathcal{M}$ starts simulating a call of M_ϕ given input v. This simulation will use at most polynomial space and the answer returned by M_ϕ is $\mathbf{1}$ if and only if $i_s \models \phi$. So $\mathcal{M}$ solves the model checking problem.

It remains to argue for the complexity of $\mathcal{M}$. Any states of s can be stored in polynomial space. $\mathcal{M}_\mathcal{C}$ and T_{η_s} are polynomial bounded space machines. which can be simulated by $\mathcal{M}$ in polynomial space. Properties about (V_s, E_s) can be decided in polynomial space, see the proof of Theorem 9. The subformula machines can be constructed in polynomial time and each of them use at most polynomial space. When simulating the subformula machines, $\mathcal{M}$ only needs a stack of depth at most $|\phi|$, where for each waiting subroutine call, $\mathcal{M}$ only needs a polynomial amount of space. We therefore conclude that $\mathcal{M}$ solves the model checking problem for $\mathsf{B}(X, U)$ using only polynomial space. □

4 Example of an Application

In this section we define K-bounded nets and describe how they can be specified as a class of compact systems.

Definition 11. A *K-bounded Place/ Transition net*, or just a *K-bounded net*, is a tuple $N = (P, T, F, M_{init})_K$ such that

- P and T are finite disjoint nonempty sets; their elements are called *places* and *transitions*, respectively.
- $F \subseteq (P \times T) \cup (T \times P)$; F is called the *flow relation.*
- $M_{init}: P \to \{0, 1, 2 \ldots, K\} \subseteq I\!N$; M_{init} is called the *initial marking* of N; in general, a mapping $M: P \to \{0, 1, 2 \ldots, K\} \subseteq I\!N$ is called a *marking* of N. We shall use the notation $p \in M$ if $M(p) > 0$.

Next, we define the behaviour of K-bounded Petri nets.

Definition 12. Given a K-bounded net $N = (P, T, F, M_{init})_K$.

- A transition $t \in T$ is *enabled* at a marking M of N if $M(p) > 0$ for every place p in ${}^\bullet t = \{p \mid (p,t) \in F\}$, the preset of t, and $M(p) < K$ for every place p in $t^\bullet \backslash {}^\bullet t$, where $t^\bullet = \{p \mid (t,p) \in F\}$, the postset of t. Henceforth, we shall assume that there are no isolated elements, i.e. $(\forall p \in P. {}^\bullet p^\bullet \neq)$ and $(\forall t \in T. {}^\bullet t^\bullet \neq)$.
- Given a transition t, we define a relation $\xrightarrow{t}$ between markings as follows: $M \xrightarrow{t} M'$ if t is enabled at M and $M'(s) = M(s) + F(t,s) - F(s,t)$, where $F(x,y)$ is 1 if $(x,y) \in F$ and 0 otherwise. The transition t is said to *occur* (or *fire*) at M. A marking M is a *deadlock*, denoted $M \not\rightarrow$, if it enables no transitions.
- If $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} M_n$ for some markings $M_0, M_1, \ldots M_n$, then the sequence $\sigma = t_1 \ldots t_n$ is called an *occurrence sequence* from M_0. M_n is the marking *reached* by σ, and this is denoted $M_0 \xrightarrow{\sigma} M_n$. In general, we use the notation σ for a finite or infinite sequence of transitions and use the notation $M_0 \xrightarrow{\sigma}$ to indicate that all finite prefixes of σ are occurrence sequences from M_0. Sometimes, the notation $M_0 \xrightarrow{\sigma}$ is also used to denote the sequence $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_1} \cdots$.
- An occurrence sequence σ from a marking M_0 is maximal if it is either infinite or it is finite and reaches a deadlock.
- A marking M is *reachable* from M_0 if it is the marking reached by some occurrence sequence from M_0. $[M_0\rangle$ will denote the set of markings reachable from M_0. $[M_{init}\rangle$ is the set of reachable markings of N.
- The reachability graph of N is the edge-labelled graph, (V_N, E_N), whose vertices are the reachable markings of N; if $M \xrightarrow{t} M'$ for a reachable marking M, then there is an edge from M to M' labelled t. Notice that N has at most $(K+1)^{|P|}$ markings.

Encoding a K-bounded net can be done along the lines of e.g. [HU79] Chap. 8.3. We choose the following encoding: The string s_N encodes (in binary) the number of places, the number of transitions, the pairs in F, and the initial marking. Without loss of generality, interpreting the atomic propositions $\mathcal{A}$ as places of K-bounded gives us a valuation η for all nets. η maps a pair consisting of an atomic proposition a and a place p to $\mathbf{1}$ if and only if the encoding of a equals the encoding of p. We therefore only need one T_η for the class of K-bounded Petri nets. Having one for each N would also be possible.

The machine $\mathcal{M}_\mathcal{C}$ will do the following: First, it checks that the input string encodes a K-bounded net. Assume that the net described by the input has n places. Since N is assumed to have no isolated elements, the length of the input is at least n. Hence, $\mathcal{M}_\mathcal{C}$ may then use $n \log K$ tape squares to store a marking (notice K is fixed for the class of K-bounded nets). So, it starts by storing the initial marking of the net. It then enters its signal state to signal that the configuration is a marking of the net. Then, it leaves the signal state, guesses a transition, and checks if it is enabled. If it is, $\mathcal{M}_\mathcal{C}$ "fires" it by updating the stored marking accordingly. Having done that $\mathcal{M}_\mathcal{C}$ enters the signal state, signaling that it has computed a new marking of the net. Then it continues as before; leaving the signal state, guessing a new transition to fire et.c. If the guessed transition is not enabled, $\mathcal{M}_\mathcal{C}$ just halts. Notice that if s_N is the encoding of the K-bounded net N, then the state space of s_N is isomorphic to the reachability graph of N such that i_{s_N} corresponds to the initial marking of N.

Hence K-bounded Petri nets can be specified as a class of compact systems.

Theorem 13. *The model checking problems for K-bounded Petri nets and the logics $\mathsf{L}(F)$, $\mathsf{L}(X, U, S)$, and $\mathsf{B}(X, U)$ are PSPACE-complete.*

Proof. SKETCH: The PSPACE-hardness of the problem follows from the fact that the logics can express the reachability of a marking M in a 1-safe net [CEP93] (A 1-safe net has the property that it is K-bounded for any $K \geq 1$). This can be done using an obvious formula linear in the size of P: $\phi \equiv F((\bigwedge_{p \in M} p) \wedge (\bigwedge_{p \notin M} \neg p))$. Actually, along the lines in [CEP93] one can prove that for a given 1-safe net N and a place p of N, the problem of deciding whether or not there exists a reachable marking M such that $p \in M$ $(M(p) = 1)$ is PSPACE-complete. Since this is expressed by the formula $F(p)$, model checking any reasonable propositional branching time temporal logic must be PSPACE-hard.

Since $\mathsf{L}(F)$ is a fragment of $\mathsf{L}(X, U, S)$ ($F(\phi)$ is short for $t\, U \phi$), Theorem 9 and Theorem 10 give us the matching PSPACE upper bounds. □

5 Conclusion

We have provided algorithms which give us an upper bound on the complexity of the model checking problem for a well known class of basic temporal logics interpreted over any class of compact systems, i.e. any class of systems satisfying certain conditions which limits the "succinctness" of their description; their associated state graphs must at most be exponentially larger than the models themselves.

Our results gave an upper bound for both $\mathsf{L}(X, U, S)$ and CTL. As an application of our results, we showed in Sect. 4 that the model checking problems for $\mathsf{L}(\mathsf{F})$, $\mathsf{L}(X, U, S)$, and $\mathsf{B}(X, U)$ over K-bounded nets are PSPACE-complete.

For a net N known to be bounded, that is one only knows that there exists a K such that N is K-bounded, storing any reachable marking would require exponential space in the worst case. We claim similar lower bounds hold for other nontrivial classes of compact systems. In general, we cannot give a PSPACE-hardness lower bound for a class of compact systems; R-structures are a trivial class of compact systems which doesn't have this lower bound for e.g. $\mathsf{L}(F)$.

Future research should investigate temporal logics containing and combining other operators than the ones considered here. Also, the same approach is applicable if e.g. compact systems are defined such that $\mathcal{M}_\mathcal{C}$ is exponential space bounded, i.e. the systems s describe a double exponentially large state space. One would then get EXPSPACE upper bounds.

References

[BCM+92] J. R. Bruch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[CEP93] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. In *Proc. FST&TCS 13, Thirteenth Conference on the Foundations of Software Technology & Theoretical Computer Science*, pages 326–337. Springer-Verlag (*LNCS* 761), Bombay, India, December 1993. To appear in TCS, volume 148.

[CES86] Edmund M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent system using temporal logic. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[Eme90] E. Allen Emerson. *Temporal and Modal Logic*, chapter 16. Elsevier Science Publishers, 1990. in Handbook Of Theoretical Computer Science, editor J. van Leeuwen.

[ES92] Javier Esparza and Manuel Silva. A polynomial-time algorithm to decide liveness of bounded free choice nets. *Theoretical Computer Science*, 102:185–205, 1992.

[Esp93] Javier Esparza. Model checking using net unfoldings. In *Proc. TAP-SOFT'93*, pages 613–628. Springer-Verlag (*LNCS* 668), 1993. Full version to appear in Science of Computer Programming.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

[JLL77] Neil D. Jones, Lawrence H. Landweber, and Y. Edmund Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4:277–299, 1977.

[Lam80] Leslie Lamport. "Not never" – on the Temporal Logic of programs. *Proc. 7th Ann. ACM Symp. on Princples of Programming Languages*, pages 174–185, 1980.

[Lar88] Kim G. Larsen. Proof systems for Hennessy-Milner logic with recursion. In *Proceedings of CAAP, Nancy France*, pages 215–230. Springer-Verlag (*LNCS* 299), March 1988.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice Hall International Series In Computer Science, C. A. R. Hoare series editor, 1989.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of Propositional Linear Temporal Logic. *Journal of the ACM*, 32(3):733–749, 1985.

[SW89] Colin P. Stirling and David Walker. Local model checking in the modal mu-calculus. Technical Report ECS–LFCS–89–78, Laboratory for Foundations of Computer Science, Department of Computer Science – University of Edinburgh, May 1989.

[Val90] Antti Valmari. Stubborn sets for reduced state space generation. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1990*, pages 491–515. Springer-Verlag (*LNCS* 483), 1990.

[WG93] Pierre Wolper and Patrice Godefroid. Partial-order methods for temporal verification. Technical report, Université de Liège, Institut Montefiore, August 1993. Wolper-1, Hand-outs at Summerschool in Logical Methods In Concurrency Aarhus'93, To appear in: Concur'93 Proceedings.

Recent Publications in the BRICS Report Series

RS-95-18 **Allan Cheng.** ***Complexity Results for Model Checking.*** **February 1995. 18pp.**

RS-95-17 **Jari Koistinen, Nils Klarlund, and Michael I. Schwartzbach.** ***Design Architectures through Category Constraints.*** **February 1995. 19 pp.**

RS-95-16 **Dany Breslauer and Ramesh Hariharan.** ***Optimal Parallel Construction of Minimal Suffix and Factor Automata.*** **February 1995. 9 pp.**

RS-95-15 **Devdatt P. Dubhashi, Grammati E. Pantziou, Paul G. Spirakis, and Christos D. Zaroliagis.** ***The Fourth Moment in Luby's Distribution.*** **February 1995. 10 pp.**

RS-95-14 **Devdatt P. Dubhashi.** ***Inclusion–Exclusion*(3) *Implies Inclusion–Exclusion*(n).** **February 1995. 6 pp.**

RS-95-13 **Torben Braüner.** ***The Girard Translation Extended with Recursion.*** **1995. Full version of paper to appear in Proceedings of CSL '94, LNCS.**

RS-95-12 **Gerth Stølting Brodal.** ***Fast Meldable Priority Queues.*** **February 1995. 12 pp.**

RS-95-11 **Alberto Apostolico and Dany Breslauer.** ***An Optimal*** $O(\log\log n)$ ***Time Parallel Algorithm for Detecting all Squares in a String.*** **February 1995. 18 pp. To appear in SIAM Journal on Computing.**

RS-95-10 **Dany Breslauer and Devdatt P. Dubhashi.** ***Transforming Comparison Model Lower Bounds to the Parallel-Random-Access-Machine.*** **February 1995. 11 pp.**

RS-95-9 **Lars R. Knudsen.** ***Partial and Higher Order Differentials and Applications to the DES.*** **February 1995. 24 pp.**

RS-95-8 **Ole I. Hougaard, Michael I. Schwartzbach, and Hosein Askari.** ***Type Inference of Turbo Pascal.*** **February 1995. 19 pp.**

RS-95-7 **David A. Basin and Nils Klarlund.** ***Hardware Verification using Monadic Second-Order Logic.*** **January 1995. 13 pp.**