The size of BDDs and other data structures in temporal logics model checking

Andrea Ferrara, Paolo Liberatore, and Marco Schaerf
DIAG, Sapienza University of Rome,
Via Ariosto 25, 00185 Roma, Italia.
Email: lastname@dis.uniroma1.it

***Abstract*—Temporal Logic Model Checking is a verification method in which we describe a system, the model, and then we verify whether important properties, expressed in a temporal logic formula, hold in the system. Many Model Checking tools employ BDDs or some other data structure to represent sets of states. It has been empirically observed that the BDDs used in these algorithms may grow exponentially as the model and formula increase in size. We *formally* prove that no kind of data structure of polynomial size can represent the set of valid initial states for all models and all formulae. This result holds for all data structures where a state can be checked in polynomial time. Therefore, it holds not only for all types of BDDs regardless of variable ordering, but also for more powerful data structures, such as RBCs, MTBDDs, ADDs and SDDs. Thus, the size explosion of BDDs is not a limit of these specific data representation structures, but is unavoidable: every formalism used in the same way would lead to an exponential size blow up.**

Keywords: Model Checking, Complexity, Compilability, Succinctness, BDDs.

I. Introduction

Temporal Logic Model Checking [1] is a verification method for discrete systems. In a nutshell, the system, often called the model, is described by the possible transitions of its components, while the properties to verify are encoded in a temporal modal logic. It is used, for example, for the verification of protocols and hardware circuits [2]. Many tools, called *model checkers*, have been developed to this aim. The most famous ones are SPIN [3], SMV [4] (with its well known implementation NuSMV [5]) and PRISM [6].

Many languages can be used to express the model; the most widespread ones are Promela and SMV. Two temporal logics are mainly used to define the specification: CTL [1] and LTL [7]. A detailed survey of complexity results on temporal logic model checking can be found in [8].

In many cases, the inputs of model checking problems can be processed in different ways. As an example, if the same system (encoded as a model) has to be checked against different properties (each expressed by a formula), it makes sense to spend more time on the system if it allows checking each property more efficiently. The converse applies when the same property has to be checked against different systems. A previous article by the same authors of the present one [9] analyzes these cases.

In this article, we consider the *initial state problem*: given a formula, a model and a state, is there a run of the model that starts from the given state and that satisfies the formula? We prove that complexity does not decrease even if both model and formula undergo a preprocessing step that produces a polynomially-sized result. In a way, this shows that the choice of the initial state can encode the whole complexity of model checking.

This result allows answering a long-time standing question in Symbolic Model Checking [4], [10]. It has been observed that the BDDs [11], [12], [13], [14] used by SMV and other Symbolic Model Checking systems may become exponentially large. This article formally proves that this phenomenon is not limited to BDDs, but holds for every possible form of representation of sets of states in which a state can be checked for membership in polynomial time.

Some specific Boolean functions have already been shown not to be representable with polynomial-size BDDs [15]. The results in this article proves that this is the case for every data

structure of this kind aimed at representing sets of states in symbolic model checking. It holds for all decision diagrams representing integer-value functions whose evaluation problem is in the polynomial hierarchy. Therefore it also applies to all current, and future, enhancements of BDDs, such as BMD and *BMD [16], RBCs [17], MTBDDs [18], ADDs [19] and SDD [20].

II. Definitions

In this section we briefly introduce the terminology and important properties of LTL and CTL temporal model checking [1], [7] and automated planning formalism STRIPS [21]. For more details we refer the reader to the original articles.

A. Model checking

In this section, we briefly recall the basic definitions about model checking that are needed in the rest of the paper. We follow the notation used by Sistla and Clarke [22]. Only CTL formulae $EF\phi$ (or equivalently, LTL formulae $F\phi$) are used in this article, meaning that the propositional formula ϕ is true in some future time point. The meaning and formal definition of the other operators can be found in any of a number of articles on the topic [1], [7].

The semantics is based on Kripke structures. Given a set of atomic propositions, a Kripke structure is a tuple $\langle Q, R, \ell, I\rangle$, where Q is a set of states, R is a binary relation over states (the transition relation), ℓ is a function from states to atomic propositions (it labels each state with the atomic propositions that are true in that state), I is a set of the possible initial states. A run of a Kripke structure is called a Kripke model: an infinite sequence of states such that $s_i R s_{i+1}$ for each pair of consecutive states s_i, s_{i+1} in the sequence. The truth of formulas in such a sequence $[s_0, s_1, \ldots]$ is defined as: atomic propositions are evaluated according to the initial state of the sequence (in this case $\ell(s_0)$); non-modal connectives are evaluated as usual; $F\phi$ is true if there exists i such that ϕ is true in $[s_i, s_{i+1} \ldots]$. A formula $EF\phi$ is true in a Kripke structure, which may have several Kripke models (runs), if ϕ is true in at least one of them.

In formal verification, the behavior of a system is represented as a Kripke structure, and the property to check as a modal formula. Checking the structure against the formula tells whether the system satisfies the property. Since Kripke structure is usually called a "model" (which is in fact very different from a Kripke model, which is only a possible run), this problem is called Model Checking.

In formal verification, the behavior of a system is represented as a Kripke structure, and the property to check as a modal formula. Checking the structure against the formula tells whether the system satisfies the property. Since a Kripke structure is usually called a "model" (which is in fact very different from a Kripke model, which is only a possible run), this problem is called Model Checking.

In most applications, the Kripke structure is not specified by a tuple $\langle Q, R, \ell, I\rangle$ but in a more compact form. More precisely, the set of states Q is the set of all possible interpretations over a given alphabet of variables V and the initial states and the allowed transitions between states are specified by formulas L and ϱ; the latter is a formula over variables $V \cup V'$, where V' is a new set of variables in one-to-one relation with V; These variables represent the state after the change.

Such a triple (V, L, ϱ) is called a model, and represents a Kripke structure $\langle Q, R, \ell, I\rangle$ in which Q is the set of all possible interpretations over V, ℓ is the identity function, I is the set of interpretations satisfying L, and sRs' holds if the formula ϱ is satisfied by the model $s \cup s'[V/V']$ ($s \cup s'[V/V'] \models \varrho$), where $s'[V/V']$ denotes the state obtained by substituting each variable $x \in V$ with its corresponding variable $x' \in V'$.

As a result, states are interpretations over V, and a Kripke model is an infinite sequence of states such that the first one satisfies L and each consecutive pair satisfies ϱ; this sequence is also called a run of the model.

Problems like reachability (where the formula is $EF\phi$) can be checked in time polynomial in the size of the Kripke structure, but a model can represent a Kripke structure of size exponentially larger than it. In particular, a model can have an exponential number of initial states satisfying $F\phi$, but this alone does not rule out the possibility that the set of such states can be represented in a compact form. As a limit example, $EF(p_1 \vee \cdots \vee p_n)$

holds in an exponential number of initial states in the model $\bigwedge p_i \equiv p'_i$, yet the formula $p_1 \vee \ldots \vee p_n$ itself provides a very compact representation of the set of such states.

Usually, models are specified as compositions of modules. Each module is a sort of "mini-model": it is like a model but some variables are local to it while the others are shared with the other modules. If V^L is the set of local variables of a module and V^S the set of shared variables, which is common to all modules, then the module is $M = (V^L \cup V^S, L, \varrho)$. A model can be then obtained as the composition of a number of modules, each set of local variables being disjoint from all others.

A set of modules $M_1, \ldots, M_k$ can be composed, forming a single model, by using the following set of variables and initial state formula:

$$\begin{aligned} V &= V^S \cup \bigcup_{i=1,\ldots,k} V_i^L \\ L &= \bigwedge_{i=1,\ldots,k} L_i \end{aligned}$$

The transition formulas can be composed according to a synchronous or an (interleaved) asynchronous behavior. In the synchronous composition, every module changes state at every time point: $\varrho = \bigwedge_{i=1,\ldots,k} \varrho_i$.

In the interleaved asynchronous composition, only one module at time can change state. There is however no specified rule as for which module changes at each time, so for example a module can change state three times in a row. This is formalized as follows:

$$\varrho = \bigvee_{i=1,\ldots,k} \left[\varrho_i \wedge \bigwedge_{j=1,\ldots,i-1,i+1,\ldots,k} V_j^L \equiv V_j'^L \right] \quad (1)$$

This formula is satisfied if one of its subformulas is satisfied. Therefore, when the subformula of index i changes state it forces all local variables of modules $M_1, \ldots, M_{i-1}, M_{i+1}, \ldots, M_k$ to maintain their value, while the local variables of module M_i and the shared variables change value as specified by ϱ_i.

Definition 1 (Initial state problem): Given a model M, a formula ϕ and an initial state, is there a run of M starting from that state and satisfying ϕ in some state?

This problem can be recast into the existence of a propositional formula I satisfied by exactly one propositional interpretation and such that $I \wedge EF\phi$ holds in the model.

The complexity of temporal logic model checking has been extensively studied in the literature and a good survey can be found in [8].

B. STRIPS

We use the STRIPS planning formalism [21] as the source problem for our reduction, because of the structural similarities among STRIPS planning and model checking.

A STRIPS instance (or STRIPS domain) is a 4-tuple $\langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{P}$ is the set of conditions, $\mathcal{O}$ is the set of operators, $\mathcal{I}$ is the initial state, and $\mathcal{G}$ is the goal. The *conditions* are facts that can be true or false. A *state* s is a set of conditions, and represents the state of the world at a certain time point: conditions in s represent facts that are true in the world, those not in s represent facts currently false.

The *initial state* is a state, thus a set of conditions. The *goal* is specified by a set of conditions that should be achieved, and another set specifying which conditions should not be made true. Thus, a goal $\mathcal{G}$ is a pair $\langle \mathcal{M}, \mathcal{N} \rangle$, where $\mathcal{M}$ is the set of conditions that should be made true and $\mathcal{N}$ is the set of conditions that should be made false.

The *operators* are actions that can be performed to achieve the goal. Each operator is a 4-tuple $\langle \phi, \eta, \alpha, \beta \rangle$, where ϕ, η, α, and β are sets of conditions. An operator can be executed if and only if the conditions in ϕ are true and those in η are false, and its execution makes the conditions in α true, and those in β false. The conditions in ϕ and η are called the positive and negative *preconditions* of the operator. The conditions in α and β are called the positive and negative *effects* or *postconditions* of the operator. Given a state s and an operator $o = \langle \phi, \eta, \alpha, \beta \rangle$, $r(s, o)$ denotes the result of applying action o in state s and is defined as follows:

$$r(s, \langle \phi, \eta, \alpha, \beta \rangle) = \begin{cases} (s \cup \alpha) \backslash \beta \\ \quad \text{if } \phi \subseteq s \text{ and } \eta \cap s = \emptyset \\ \text{undefined} \quad \text{otherwise} \end{cases}$$

A plan for $\langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is a sequence of operators $o_1, o_2, \ldots, o_k$ such that $s = r(r(\ldots, r(\mathcal{I}, o_1)), \ldots), o_{i-1}), o_i)$ is defined for all $i \in \{1, \ldots, k\}$ and it holds that $\mathcal{M} \subseteq s$ and $\mathcal{N} \cap s = \emptyset$.

The *plan-existence problem* for STRIPS can be formulated as follows: Given a planning instance $\langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, decide whether it has a plan. For a more detailed presentation of STRIPS and its properties we refer the reader to the article by Fikes and Nilsson [21]. In this article, we consider the restrictions STRIPS_k where the goal is composed of a single positive condition and, for each operator, the maximum number of preconditions is k and there are at most 2 postconditions. Plan existence is PSPACE-hard for every $k \geq 2$, as proved by Bylander [23].

III. Complexity and Compilability

This article employs the complexity class PSPACE, which is the set of decision problems that can be solved in space polynomial in the size of the input [24], [25]. The class P is defined in the same way with polynomial time instead of space. The polynomial hierarchy [26] is a hierarchy of classes of decision problems whose basic class is P and it is completely included in PSPACE. It is generally assumed to be a proper hierarchy, that is for each k, level k is strictly included in level $k + 1$. The collapse of the polynomial hierarchy is considered very unlikely and many results in computational complexity are frequently stated as "The property holds unless the polynomial hierarchy collapse", such in [27]. We assume that the instances of a decision problem are strings built over a finite arbitrary alphabet Σ. The set of all strings over Σ is denoted by Σ^*. The *length* of a string $x \in \Sigma^*$ is denoted by $||x||$. A language (decision problem) is defined as a set of strings.

We recall some definitions and results about the on-line complexity of problems [28]. This deals with problems that, as many do, have instances that can be naturally broken into two parts: one part known in advance and one part known only at run-time. An instance of such problems can be represented as a *pair of strings* rather than a single string. Therefore, the language corresponding to these decision problems is a language over pairs of strings, i.e., a set of pairs of strings (rather than a set of strings). Formally, a *language of pairs* is a subset of $\Sigma^* \times \Sigma^*$, where $\times$ is the Cartesian product.

The difference between the first and second string of a pair is that the first is known in advance, meaning that more time can be spent on it. This could be useful to solve the problem faster when the second string comes to be known. The idea is that an algorithm can take advantage of the difference between the first and the second part of the input:

preprocessing phase:
: elaborates only the first part of the input; since that part is known advance, this phase may take a long time;

online processing:
: when the rest of the input arrives, the algorithm can use both it and the result of the preprocessing phase to solve the problem.

While the first phase is allowed to take more time than the second one, some constraints have to be put on it: we impose its result to be of polynomial size. *Poly-size* functions are introduced to this purpose: a function f from tuples of strings to strings is called *poly-size* if there exists a polynomial p such that, for all tuples of strings $\langle x_1, \ldots, x_n \rangle$, the inequality $||f(\langle x_1, \ldots, x_n \rangle)|| \leq p(||x_1|| + \cdots + ||x_n||)$ holds. We extend this definition to mixed tuples of strings and integers by considering each number i as a string of length i; more details are in the article where poly-size functions have been introduced [28].

In the computational complexity framework, every function computable in polynomial time is also poly-size [29], but not vice versa. For example, the function that computes whether there exists a winning strategy for a generalized N by N version of checkers requires exponential time [30] to be computed but its output is either "yes" or "no". In spite of the long computing time, the produced value is always representable with a single bit.

This article uses classes that characterize the complexity of problems that can be expressed as languages of pairs, where the first input (the fixed part) is available in advance and the second one (the varying part) is only available at the last moment. The idea is that a class such as

$\|\!\rightsquigarrow$P contains all problems that are in P *after preprocessing the first part of the data*, that is, neglecting the complexity of the preprocessing phase, and concentrating only on the complexity of what remains to be done after the preprocessing step. We constraint the preprocessing phase to produce a polynomially sized result.

In a similar way, the class $\|\!\rightsquigarrow$PSPACE contains all problems that can be solved in polynomial space after the preprocessing phase is over. For technical reasons, we make the assumption that preprocessing can use not only the fixed part of the instance but also the size of the varying part. This will not be a problem because the impossibility of making a problem belong in a class will carry on to the case in which this size is not given [28].

Definition 2: The class $\|\!\rightsquigarrow$PSPACE contains all languages of pairs $S \subseteq \Sigma^* \times \Sigma^*$ such that there exist a poly-size function f from strings and integers to strings and another language of pairs $S' \in$ PSPACE such that, for all $\langle x, y\rangle \in S$, the following holds:

$$\langle x, y\rangle \in S \quad \text{iff} \quad \langle f(x, ||y||), y\rangle \in S'$$

The poly-size function f of this definition represents the preprocessing phase: $f(x, ||y||)$ is the result of preprocessing the first element of the data plus the size of the second element. Since S' is in PSPACE, the problem is in PSPACE given the result $f(x, ||y||)$ of preprocessing. The preprocessing phase is allowed to know, in addition to x, the size of y (i.e., $||y||$) for technical reasons [28].

Every problem in PSPACE is also in $\|\!\rightsquigarrow$PSPACE. Proving that a problem is also one of the most difficult ones in $\|\!\rightsquigarrow$PSPACE cannot be done with the usual definition of polynomial reduction. A new definition of reduction is necessary.

Definition 3: A *nucomp reduction* is a triple of functions $\langle f_1, f_2, g\rangle$, where g is poly-time and f_1 and f_2 are poly-size. A problem A is *non-uniformly comp-reducible* to a problem B (denoted $A \leq_{nucomp} B$) iff there exists a non-uniform comp-reduction $\langle f_1, f_2, g\rangle$ such that, for every pair $\langle x, y\rangle$, the following holds:

$$\langle x, y\rangle \in A \text{ if and only if } \langle f_1(x, ||y||), g(f_2(x, ||y||), y)\rangle \in B$$

The rationale behind this formal definition is quite technical, but it intuitively means that the fixed part of A is treated differently than the varying part. These reductions allows for a concept of *hardness* and *completeness*. More details and justifications can be found in [28].

Definition 4: A language of pairs S $\|\!\rightsquigarrow$PSPACE-*hard* iff for all problems $A \in \|\!\rightsquigarrow$PSPACE, it is the case that $A \leq_{nucomp} S$. Moreover, S is $\|\!\rightsquigarrow$PSPACE-complete if S is in $\|\!\rightsquigarrow$PSPACE and is $\|\!\rightsquigarrow$PSPACE-hard.

It can be proved that if a $\|\!\rightsquigarrow$PSPACE-hard problem could be solved in polynomial time after preprocessing the fixed part, and this preprocessing phase obeys the above restriction of producing a polynomially-sized result, then the polynomial hierarchy collapses to its second level. The details of this result can be found in other articles [28], [31], but are unnecessary for the aim of this article; what matters is that, as already discussed, such a collapse is currently considered very unlikely by the computational complexity community.

We show a technique to derive a nucomp reduction from a (regular) poly-time reduction. More precisely, we present sufficient conditions allowing for a polynomial-time reduction to imply the existence of a nucomp reduction [31].

Let us assume that we know a polynomial reduction from problem A to problem B, and we want to prove the nucomp-hardness of B. This can be accomplished by first proving some conditions on A, and then one condition on the reduction from A to B.

Definition 5: A *classification function* for a problem A is a polynomial time function $Class$ from instances of A to nonnegative integers, such that $Class(y) \leq ||y||$.

Typically, the classification function computes the number of "objects" the instance y is build upon. For example, if A is a problem on graphs, $Class(y)$ could be the number of nodes of the graph y. If A is a problem on propositional formulas, $Class(y)$ could be the number of variables in the formula y.

Definition 6: A representative function for a problem A is a polynomial time function $Repr$ from nonnegative integers to instances of A such that for every integer $n \geq 0$ it is the case that $Class(Repr(n)) = n$ and $||Repr(n)||$ is bounded

by a polynomial in n.

The representative function simply computes an element of a class. As an example, for problems of graphs the representative of the class n may be the graph with n nodes and no edge.

Definition 7: An *extension function* for a problem A is a polynomial time function Ext from instances of A and nonnegative integers to instances of A such that, for each y and $n \geq Class(y)$, the instance $y' = Ext(y, n)$ satisfies the following conditions:

1) $y \in A$ if and only if $y' \in A$;
2) $Class(y') = n$.

An extension function increases the class of an instance without changing its membership to the language.

Let us give some intuition about these functions. Usually, an instance of a problem is composed of a set of objects combined in some way. For problems on Boolean formulas, we have a set of variables combined to form a formula. For graph problems, we have a set of nodes, and the graph is indeed a set of edges, which are pairs of nodes. The classification function gives the number of objects in an instance. The representative function thus gives an instance composed of the given number of objects. This instance should be in some way "symmetric", in the sense that its elements should be interchangeable (this is because the representative function must be determined only from the number of objects.) Possible results of the representative function can be the set of all clauses of three literals over a given alphabet, the complete graph over a set of nodes, the graph with no edges, etc.

Let, as an example, A be the problem of propositional satisfiability. $Class(F)$ could be the number of variables in formula F and $Repr(n)$ the set of all clauses of three literals over an alphabet of n variables. A possible extension function simply adds a suitable number of tautological clauses.

These three functions are related to the problem A only, and involve neither the specific problem B we want to prove hard nor the specific reduction used. We now define a condition over the poly-time reduction from A to B. Since B is a problem of pairs, we can define a reduction from A to B as a pair of polynomial functions $\langle r, h\rangle$ such that $x \in A$ if and only if $\langle r(x), h(x)\rangle \in B$.

Definition 8: Given a problem A with associated $Class$, $Repr$ and Ext functions, a problem of pairs B, and a polynomial reduction $\langle r, h\rangle$ from A to B, the condition of representative equivalence holds if, for any instance y of A, the following holds:

$$\langle r(y), h(y)\rangle \in B \quad \text{iff}$$
$$\langle r(Repr(Class(y)), h(y)\rangle \in B$$

Representative equivalence requires an extension function Ext to exists, even if it is not mentioned in the "iff" condition. It can be shown that the existence of such a reduction from a PSPACE-hard problem A to a problem B proves that the latter is $\|\!\rightarrow$PSPACE-hard [31].

IV. Planning

A preliminary step to prove the claim about model checking in temporal logic is to show that a certain restriction of the planning problem STRIPS is $\|\!\rightarrow$PSPACE-hard.

Theorem 1: The problem of plan-existence in STRIPS_3 is $\|\!\rightarrow$PSPACE-hard if the varying part of an instance is the initial state.

Proof: We show a reduction from STRIPS_2 to STRIPS_3 satisfying the condition of representative equivalence. It would had been formally more elegant to reduce STRIPS_2 to STRIPS_2 or to reduce STRIPS_3 to STRIPS_3, but to satisfy the condition of representative equivalence we need to introduce a precondition to each operator, thereby turning a STRIPS_2 instance into a STRIPS_3 one.

As previously mentioned, the problem of plan existence for STRIPS_2 is PSPACE-hard. Its required three functions are:

Classification:
$$Class(\langle\{p_1, \ldots, p_n\}, \mathcal{O}, \mathcal{I}, \mathcal{G}\rangle) = n$$
Representative:
$$Repr(n) = \langle\{p_1, \ldots, p_n\}, O_n^{2/2}, \emptyset, \langle\{p_1\}, \emptyset\rangle\rangle,$$
where $O_n^{2/2}$ is the set of all possible operators of at most two preconditions and two postconditions of $\{p_1, \ldots, p_n\}$
Extension:
$$Ext(\langle\{p_1, \ldots, p_m\}, \mathcal{O}, \mathcal{I}, \mathcal{G}\rangle, n) = \langle\{p_1, \ldots, p_m, p_{m+1}, \ldots, p_n\}, \mathcal{O}, \mathcal{I}, \mathcal{G}\rangle$$

The reduction produces an instance of STRIPS_3 by suitably modifying a STRIPS_2 instance. In particular, new conditions are created and inserted in the initial state, and each operator is modified individually. For each operator $o = \langle \alpha, \beta, \gamma, \delta \rangle$ of the original instance, a condition p_o is added to the set of conditions and to the initial state, and o is changed into $o' = \langle \alpha \cup \{p_o\}, \beta, \gamma, \delta \rangle$; this is a valid STRIPS_3 instance because $||\alpha|| \leq 2$ and $||\beta|| \leq 2$ since the original instance is of STRIPS_2, therefore $||\alpha \cup \{p_o\}|| \leq 3$. The goal is not changed. In this new instance, no operator modifies p_o, which therefore always remains true. As a result, each operator o' is executable if and only if o is. Therefore, a plan exists for this new instance if and only if one exists for the original instance.

We prove that this reduction satisfies the condition of representative equivalence. Let y be an instance of STRIPS_2 and $\langle r(y), h(y) \rangle$ its corresponding STRIPS_3 instance, where $h(y)$ is the initial state and $r(y)$ the rest of the instance. We show that $\langle r(y), h(y) \rangle$ has the same plans of $\langle r(z), h(y) \rangle$, where $z = Repr(Class(y))$.

The two instances y and z have the same conditions and goal, but differ on the initial state and operators. In particular, z has every operator with two preconditions and two postconditions, while y in general may contain only some of them. The corresponding instances $\langle r(y), h(y) \rangle$ and $\langle r(z), h(y) \rangle$ therefore only differ for their operators, since the initial state $h(y)$ is the same in them.

In particular, $r(y)$ contains an operator $o' = \langle \alpha \cup \{p_o\}, \beta, \gamma, \delta \rangle$ for every operator $o = \langle \alpha, \beta, \gamma, \delta \rangle$ of y whereas $r(z)$ contains one such operator o' for each possible operator o with two preconditions and two postconditions. However, $h(y)$ contains p_o only if o is an operator of y. As a result, the condition p_o for an operator o that is in z but not in y is not in the initial state of $\langle r(z), h(y) \rangle$. Since no operator has it as a postcondition, the corresponding operator o' is never executable. As a consequence, the operators in $\langle r(z), h(y) \rangle$ that are not in $\langle r(y), h(y) \rangle$ are not part of any plan of the former instance.

In order to complete the proof, the representation of conditions and operators has to be specified. Indeed, we have to make sure that replacing $r(y)$ with $r(z)$ in $\langle r(y), h(y) \rangle$ does not create a mismatch between the representation of conditions and operators in y and z.

The chosen representation has alphabet $\{0, \ldots, 9, (,), \langle, \rangle, c\}$. A condition like p_{34} is represented simply by the two-character string 34. Similarly, the operator that has p_{34} and p_1 as positive preconditions, no negative precondition or postcondition and p_{123} as the positive postcondition, is represented by the string $\langle (34c1)()(123)() \rangle$. In such a string, (represents what is commonly written as an open curly brace {; in the same way,) means the closed curly brace and c stands for a comma.

A new condition p_o created by the reduction is represented by the same string that represents o, the distinction between condition and operator being clear from the context. For instance, the example operator of the previous paragraph generates the new condition p_o represented by the same string $\langle (34c1)()(123)() \rangle$ and the new operator o' represented by the string $\langle (34c1c\langle (34c1)()(123)() \rangle)()(123)() \rangle$.

There is no recursion in this expression: while o' contains the string representing o, no condition $p_{o'}$ is ever created.

The introduction of the new condition and the change to the operator do not superpolynomially increase size. Indeed, for each operator the new condition has the same size of the original operator, and the new operator is twice the size of the original operator.

□

Since STRIPS_k limits preconditions to be *at most* k and postconditions to be two, STRIPS_3 is a restriction of STRIPS_k for every $k \geq 3$ and of unbounded STRIPS. As a result, the $\|\rightarrow$PSPACE-hardness of STRIPS_3 implies that all these problems are $\|\rightarrow$PSPACE-hard as well.

In the opposite way, since STRIPS is in PSPACE, it is in $\|\rightarrow$PSPACE as well, as well as all its restrictions. Together with the above hardness result, this proves that STRIPS and all STRIPS_k problems with $k \geq 3$ are $\|\rightarrow$PSPACE-complete.

V. Interleaved asynchronous composition

The main aim of this section is to show that the initial state problem of model checking is

$\|\!\rightsquigarrow$PSPACE-hard when the model is represented as an asynchronous interleaved composition of modules. This is achieved by reducing STRIPS_3, proved hard for that class in the previous section, to that problem.

Theorem 2: The initial state problem is $\|\!\rightsquigarrow$PSPACE-complete, if the model is represented as an asynchronous interleaved composition of modules and the varying part of the problem is the initial state.

Proof: We show a reduction from STRIPS_3. In this reduction, the initial state is translated into the initial state, and the rest of the planning instance is translated into the model and the formula of the initial state problem.

An operator of a planning instance specifies conditions over the states before and after its execution. This can be directly translated into the formula of a model checking module. In particular, this formula is true if the previous state makes the operator executable, the effects of the operators are true in the next state and all other conditions maintain their truth value through the transition.

Let $\langle \{p_1, \ldots, p_n\}, \mathcal{O}, \mathcal{I}, \langle \{p_1\}, \emptyset \rangle \rangle$ be an instance of STRIPS_3. For each operator $o_i = \langle \alpha_i, \beta_i, \gamma_i, \delta_i \rangle$ in $\mathcal{O}$, we define a module $M_i = (V_i^L \cup V^S, \mathsf{true}, \varrho_i)$ where $V_i^L = \{n_i\}$, $V^S = \{p_1, \ldots, p_n\}$, and ϱ_i is as follows:

$$\varrho_i \;=\; (n_i \not\equiv n_i') \wedge \bigwedge_{p_j \in \alpha_i} p_j \wedge \bigwedge_{p_j \in \beta_i} \neg p_j \wedge \bigwedge_{p_j \in \gamma_i} p_j' \wedge \bigwedge_{p_j \in \delta_i} \neg p_j' \wedge \bigwedge_{p_j \notin \delta_i \cup \gamma_i} (p_j' \equiv p_j)$$

Each module represents the execution of the corresponding operator. Indeed, this formula can only be satisfied if the precondition of the operator is true in the previous state, the postcondition is true in the next state, and all other conditions maintain their truth values.

There is exactly one variable n_i for each module. The subformula $n_i \not\equiv n_i'$ ensures that if a module is "active", the others are not. Indeed, the transition function ϱ of the composition is defined in (1) as the disjunction of a subformula for each module, and each subformula forces the local variables of the other modules to maintain their value. As a result, if the subformula corresponding to an operator is true, then the local variables of all other modules will be forced to maintain their values.

The interleaved composition of the modules ϱ_i is a model that allows, at each step, only the changes resulting from executing exactly one operator of the STRIPS_3 instance: the variables n_i and their subformulae ensure that the translation is what results from applying a single operators and not two or more at the same time while the rest of the formulae M_i encode the behavior of an operator each. As a result, the existence of a plan equates the existence of a run of the model that starts from the given initial state and satisfies p_1 in some time point. This condition is the same as EFp_1 being true in the given initial state of the model.

Therefore, this is a translation from the problem of planning into the problem of checking whether the model has a run, starting from a given initial state, that satisfies a given formula. In this translation, the formula is fixed, the initial state is translated into the initial state and the rest of the instance is translated into the rest of the instance. This is therefore a nucomp reduction.

Technically, a nucomp reduction is a triple of functions f_1, f_2 and g that operate on the fixed part of the planning problem $\{p_1, \ldots, p_n\}$, $\mathcal{O}$, $\langle \{p_1\}, \emptyset \rangle$, the varying part of the planning problem $\mathcal{I}$ and its size $||\mathcal{I}||$. In particular:

$f_1(\langle \{p_1, \ldots, p_n\}, \mathcal{O}, \langle \{p_1\}, \emptyset \rangle \rangle, ||\mathcal{I}||)$
: is the pair composed of the interleaved asynchronous composition of the M_i's above and of the formula EFp_1;

$f_2(\langle \{p_1, \ldots, p_n\}, \mathcal{O}, \langle \{p_1\}, \emptyset \rangle \rangle, ||\mathcal{I}||)$
: is EFp_1;

$g(x, \mathcal{I})$
: is $\mathcal{I}$ for every x.

In this specific case f_2 is not really needed; yet, the definition of nucomp reduction dictates that one has to be provided. The choice of EFp_1 for its value is arbitrary.

We now formally prove that the reduction works as required. We assume that the planning instances always contain a "nop" operator, that is, $\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$; this is necessary to ensure that transitions are still possible in the model after the goal is reached. We denote by N the set of all variables n_i.

1. correspondence between operators and modules

Let s and o_i be respectively a state and an operator of the STRIPS_3 instance, and let $s' = r(s, o_i)$. We prove that both $\langle s \cup N_i \cup \{n_i\}, s' \cup N_i \rangle$ and $\langle s \cup N_i, s' \cup N_i \cup \{n_i\} \rangle$ are pairs of previous/next states allowed by the model, for every $N_i \subseteq N \backslash \{n_i\}$. We also prove the converse: if a pair of states is allowed by the model, then it is either in the form $\langle s \cup N_i \cup \{n_i\}, s' \cup N_i \rangle$ or $\langle s \cup N_i, s' \cup N_i \cup \{n_i\} \rangle$, where $s' = r(s, o_i)$ and N_i is a subset of $N \backslash \{n_i\}$.

Assume that $s' = r(s, o_i)$ and let $o_i = \langle \alpha, \beta, \gamma, \delta \rangle$. Since $r(s, o_i)$ is not undefined, this operator is executable in s. As a result, the formula $\bigwedge_{p_i \in \alpha} p_i \wedge \bigwedge_{p_i \in \beta} \neg p_i$ is true in the state $s \cup N_i \cup \{n_i\}$. Since s' differs from s only by the addition of γ and deletion of δ, the formula $\bigwedge_{p_i \in \gamma} p'_i \wedge \bigwedge_{p_i \in \delta} \neg p'_i \wedge \bigwedge_{p_i \notin \delta \cup \gamma} (p'_i \equiv p_i)$ holds as well in the resulting state $s' \cup N_i$. The same holds for the pair $\langle s \cup N_i, s' \cup N_i \cup \{n_i\} \rangle$.

This proves that ϱ_i is satisfied by all considered pairs. Since the values of n_j for $j \neq i$ is the same in the state before and after the transition, and these are the local variables of the other modules, the i-th disjunct of ϱ (see equation 1) is satisfied, which means that ϱ itself is satisfied as well.

Let us now prove the converse. Let s, s' be a pair of previous/next state allowed by the model. Since ϱ is true, (at least) one of its disjuncts has to be satisfied. Let it be the i-th one. We prove that $s' \backslash N = r(s \backslash N, o_i)$.

Since the i-th subformula of ϱ is satisfied, all variables N have the same value in s and s' but n_i, which has opposite values. Furthermore, ϱ_i is satisfied by the values of variables in P. This means that all variables in α are true and all those in β are false in s, that the variables in γ are true and those in δ are false in s', and that all other variables maintain their values between s and s'. This proves that $s' \backslash N = r(s \backslash N, o_i)$.

2. Correspondence between sequences

We proved that the execution of an operator corresponds to a transition allowed by the model and vice versa. By using the same initial state, the result of applying a sequence of operators is an allowed multi-step transition according to the model, if one disregards the values of the variables N_i.

As a result, the existence of a plan implies that EFp_1 holds in the initial state of the model. Since plan existence is $\|\!\rightsquigarrow$PSPACE-complete, it follows that the initial state problem is $\|\!\rightsquigarrow$PSPACE-hard. Membership to the same class follows from the initial state problem being in PSPACE. □

The result of the preprocessing phase can be considered as a representation of the set of states satisfying the initial state problem. As a consequence, the above theorem has the following result as a corollary.

Corollary 1: The polynomial hierarchy collapses if, given a formula and a model represented by the asynchronous interleaved composition of modules, there always exists a polynomially sized data structure that allows checking whether a state is a solution of the initial state problem in polynomial time.

This result has a practical implication on algorithms using the labeling technique for solving model checking. The labeling technique works by determining the set of states satisfying the subformulas of the formula, starting from the literals and proceeding in a bottom-up manner. Since the number of states is exponential in the number of variables, BDDs are used to represent these sets. The procedure ends when we generate a BDD representing the set of initial states satisfying the formula. This is exactly a data structure that allows checking whether a state is an initial state satisfying the formula in polynomial time. Therefore, the above corollary applies to this case.

Corollary 2: Unless the polynomial hierarchy collapses to its second level, the BDD representation of the initial states satisfying a formula and a model defined as an asynchronous interleaved composition of modules is, in the worst case, superpolynomial in the size of the formula and model.

This result theoretically confirms the empirical observations about the size increase of BDDs in model checking. The previous corollary however qualifies it as not being specific to BDDs, as the same holds for any other formalism with similar features. In fact, the same result also applies to RBCs, MTBDDs, SDDs, and ADDs.

VI. SYNCHRONOUS COMPOSITION

When modules are composed synchronously, all their formulas ϱ_i apply to every transition. Formally, the difference from the interleaved asynchronous composition is the way the formulas ϱ_i are composed to form ϱ.

We show a translation from asynchronous to synchronous composition that does not involve the initial state. This reduction adds a shared variable a_i to each module. A new module ϱ_0^a is created, and each module ϱ_i is translated into ϱ_i^a as follows:

$$\varrho_i^a = a_i \rightarrow (\varrho_i \wedge \bigwedge_{\substack{j=1,\ldots,k \\ j\neq i}} V_j^L \equiv V_j^{L\prime})$$

The new module ϱ_0^a enforces exactly one a_i to be true at each time step:

$$\varrho_0^a = \bigvee_{i=1,\ldots,k} \left(a_i \wedge \bigwedge_{\substack{j=1,\ldots,k \\ j\neq i}} \neg a_j \right)$$

Intuitively, this reduction works because, at any time point, at least one of the variables a_i has to be true, but no more then one. If a_i is false, then ϱ_i^a is satisfied. If a_i is true, then ϱ_i^a becomes equivalent to the i-th disjunct of ϱ, the composed formula of the asynchronous composition, as defined in (1). This means that ϱ is true if and only if the pair of states satisfies the corresponding formula for the interleaved asynchronous composition.

Theorem 3: Disregarding the variables a_i, the models that are the asynchronous interleaved composition of the $\varrho_1, \ldots, \varrho_k$ and the synchronous composition of $\varrho_0^a, \varrho_1^a \ldots, \varrho_k^a$ are the same.

Proof: We show that ϱ^a, the synchronous composition of $\varrho_0^a, \varrho_1^a \ldots, \varrho_k^a$, is essentially the same as ϱ, the asynchronous interleaved composition of the $\varrho_1, \ldots, \varrho_k$, the only difference being the values of the new variables a_i.

$$\begin{aligned}
\varrho^a &= \bigwedge_{i=0,\ldots,k} \varrho_i^a \\
&= \bigvee_{i=1,\ldots,k} \left(a_i \wedge \bigwedge_{\substack{j=1,\ldots,k \\ j\neq i}} \neg a_j \right) \wedge \\
&\quad \bigwedge_{i=1,\ldots,k} a_i \rightarrow (\varrho_i \wedge \bigwedge_{\substack{j=1,\ldots,k \\ j\neq i}} V_j^L \equiv V_j^{L\prime}) \\
&= \bigvee_{i=1,\ldots,k} \left(a_i \wedge \bigwedge_{\substack{j=1,\ldots,k \\ j\neq i}} \neg a_j \wedge \right. \\
&\quad \left. \bigwedge_{h=1,\ldots,k} a_h \rightarrow (\varrho_h \wedge \bigwedge_{\substack{j=1,\ldots,k \\ j\neq h}} V_j^L \equiv V_j^{L\prime}) \right)
\end{aligned}$$

If $h \neq i$ the formula $a_h \rightarrow \ldots$ is entailed by $\neg a_h$. As a result, ϱ^a is also equivalent to:

$$\begin{aligned}
\varrho^a &= \bigvee_{i=1,\ldots,k} \left(a_i \wedge \bigwedge_{\substack{j=1,\ldots,k \\ j\neq i}} \neg a_j \wedge \right. \\
&\quad \left. a_i \rightarrow (\varrho_i \wedge \bigwedge_{\substack{j=1,\ldots,k \\ j\neq i}} V_j^L \equiv V_j^{L\prime}) \right)
\end{aligned}$$

This formula can be further simplified:

$$\begin{aligned}
\varrho^a &= \bigvee_{i=1,\ldots,k} \left(a_i \wedge \bigwedge_{\substack{j=1,\ldots,k \\ j\neq i}} \neg a_j \wedge \right. \\
&\quad \left. (\varrho_i \wedge \bigwedge_{\substack{j=1,\ldots,k \\ j\neq i}} V_j^L \equiv V_j^{L\prime}) \right)
\end{aligned}$$

Apart from the variables a_i, this formula is equivalent to the one defining the allowed transitions of the interleaved asynchronous composition of modules ϱ_i. Formally, the variables a_i can be removed by performing the operation of *forgetting* [32]; the resulting formula is equivalent to (1). □

From the above theorem we can derive the class of the initial state problem when the model is a synchronous composition of modules.

Corollary 3: The initial state problem is $\|\!\rightarrow$PSPACE-complete if the initial state is the varying part of the instances and the model is represented as a synchronous composition of modules.

As a result, the statement of Corollary 1 holds even in the case of synchronous composition of modules: unless the polynomial hierarchy collapses, the set of the initial states of a model satisfying a property cannot in general be represented by a data structure that is polynomially sized and allows checking a state in polynomial time. As in the case of asynchronous composition of modules, this results concerns BDDs but also BRBCs, MTBDDs, SDDs, and ADDs.

VII. Discussion and related work

This article presents results about preprocessing in Temporal Logic Model Checking. These results extend, refine and improve over a previous work by the same authors [9] where the fixed part is the formula and the varying one is the model or vice versa. In particular, the present article strengthens these results by placing the initial state only in the varying part of the problem, so that the formula and most of the model are fixed. This result also appeared in a technical report [33], where the formula is however not assumed to be a composition of modules.

The results in the present article also improve over those about the exponential growth of BDDs with respect to a particular problem (e.g. integer multiplication [34]) and about the size growth of other decision diagrams with respect to particular problems [35]. While these results are not conditional to the collapse of the polynomial hierarchy as the ones reported in this paper, they are also less general, as they concern only specific data structures (e.g. OBDDs) and specific problems (e.g. integer multiplication).

Also related is the computational analysis of model checking, in particular its parametrized complexity [36]. All the considered problems, where the input is a synchronized product of k components, where k is the parameter, are shown intractable even in the parametrized case. A more detailed comparison of the differences and similarities between parametrized complexity and the classes like $\|\!\rightarrow$PSPACE is presented by Cadoli, Donini, Liberatore and Schaerf [28].

Other related results are [37]: the complexity of model checking does not decrease under the hypotheses of some structural restrictions (e.g. treewidth) in the input; although a CNF formula of bounded treewidth can be represented by an OBDD of polynomial size, the nice properties of treewidth-bounded CNF formulas are not preserved under existential quantification or unrolling, which is a basic operation of model checking algorithms.

References

[1] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 2000.

[2] I. Beer, S. B. David, D. Geist, R. Gewirtzman, and M. Yoeli, "Methodology and system for practical formal verification of reactive hardware," in *Proceedings of the 6th International Conference on Computer-Aided Verification (CAV'94)*, ser. LNCS, vol. 818. Springer, 1994, pp. 182–193.

[3] G. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23(5), pp. 279–295, 1997.

[4] K. McMillan, *Symbolic Model Checking*. Kluwer Academic, 1993.

[5] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An open-source tool for symbolic model checking," in *Proceedings of the 14th International Conference on Computer-Aided Verification (CAV'02)*, 2002.

[6] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[7] A. Pnueli, "The temporal logic of programs," in *Proceeding of the 18th IEEE Symposium on Foundations of Computer Science (FOCS'77)*, 1977, pp. 46–57.

[8] P. Schnoebelen, "The complexity of temporal logic model checking." *Advances in modal logic*, vol. 4, no. 393-436, p. 35, 2002.

[9] A. Ferrara, P. Liberatore, and M. Schaerf, "Model checking and preprocessing," in *AI*IA*, 2007, pp. 48–59.

[10] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 10^{20} states and beyond," *Inf. Comput.*, vol. 98, no. 2, pp. 142–170, 1992.

[11] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.

[12] P. Kissmann and J. Hoffmann, "BDD ordering heuristics for classical planning," *Journal of Artificial Intelligence Research*, pp. 779–804, 2014.

[13] B. Kell, A. Sabharwal, and W.-J. van Hoeve, "BDD-guided clause generation," in *Integration of AI and OR Techniques in Constraint Programming*. Springer, 2015, pp. 215–230.

[14] S. Edelkamp, P. Kissmann, and A. Torralba, "BDDs strike back (in ai planning)," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[15] Y. Breitbart, H. Hunt, and D. Rosenkrantz, "On the size of binary decision diagrams representing boolean functions," *Theoretical Computer Science*, vol. 145, no. 1, pp. 45–69, 1995.

[16] R. E. Bryant and Y.-A. Chen, "Verification of arithmetic circuits using binary moment diagrams," *STTT*, vol. 3, no. 2, pp. 137–155, 2001.

[17] P. Abdullah, P. Bjesse, and N. Een, "Symbolic reachability analysis based on SAT-solvers," in *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analisys of Systems (TACAS'00)*, 2000.

[18] E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao, "Multi terminal binary decision diagrams: An efficient data structure for matrix representation," in *Proceedings of the International Workshop on Logic and Synthesis*, 1993, pp. 1–15.

[19] R. Bahar, E. Frohm, C. Gaona, C. Hachtel, G. Macii, and F. Somenzi, "Algebraic decision diagrams and their applications," in *Proceedings of the International Conference CAD*, 1993, pp. 188–191.

[20] A. Darwiche, "SDD: A new canonical representation of propositional knowledge bases," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 1, 2011, p. 819.

[21] R. Fikes and N. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, pp. 189–208, 1971.

[22] A. Sistla and E. Clarke, "The complexity of propositional linear temporal logics," *Journal of ACM*, vol. 32(3), pp. 733–749, 1985.

[23] T. Bylander, "The computational complexity of propositional STRIPS planning," *Artificial Intelligence*, vol. 69, no. 1–2, pp. 165–204, 1994.

[24] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, Ca: W.H. Freeman and Company, 1979.

[25] D. S. Johnson, "A catalog of complexity classes," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier Science Publishers (North-Holland), Amsterdam, 1990, vol. A, ch. 2, pp. 67–161.

[26] L. J. Stockmeyer, "The polynomial-time hierarchy," *Theoretical Computer Science*, vol. 3, pp. 1–22, 1976.

[27] A. R. Klivans and D. Van Melkebeek, "Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses," *SIAM Journal on Computing*, vol. 31, no. 5, pp. 1501–1526, 2002.

[28] M. Cadoli, F. Donini, P. Liberatore, and M. Schaerf, "Preprocessing of intractable problems," *Information and Computation*, vol. 176, pp. 89–120, 2002.

[29] P. V. E. Boas, "Machine models and simulations," in *Handbook of Theoretical Computer Science, volume A*, J. Van Leeuwen, Ed. Elsevier, 2014, pp. 1–66.

[30] J. M. Robson, "N by N checkers is exptime complete," *SIAM Journal on Computing*, vol. 13, no. 2, pp. 252–267, 1984.

[31] P. Liberatore, "Monotonic reductions, representative equivalence, and compilation of intractable problems," *Journal of ACM*, vol. 48, no. 6, pp. 1091–1125, 2001.

[32] J. Lang, P. Liberatore, and P. Marquis, "Propositional independence - formula-variable independence and forgetting," *Journal of Artificial Intelligence Research*, vol. 18, pp. 391–443, 2003.

[33] A. Ferrara, P. Liberatore, and M. Schaerf, "On the size of data structures used in symbolic model checking," *Computing Research Repository (CoRR)*, vol. abs/1012.3018, 2010. [Online]. Available: http://arxiv.org/abs/1012.3018

[34] R. Bryant, "On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication," *IEEE Transactions on Computers*, vol. 40, pp. 205–213, 1991.

[35] R. Drechsler and D. Sieling, "Binary decision diagrams in theory and practice," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 3, no. 2, pp. 112–136, May 2001.

[36] S. Demri, F. Laroussinie, and P. Schnoebelen, "A parametric analysis of the state explosion problem in model checking," in *STACS*, 2002, pp. 620–631.

[37] A. Ferrara, G. Pan, and M. Y. Vardi, "Treewidth in verification: Local vs. global," in *LPAR*, 2005, pp. 489–503.