Partial-Order Methods for Model Checking: From Linear Time to Branching Time

Bernard Willems Pierre Wolper
Université de Liège
Institut Montefiore, B28
4000 Liège Sart Tilman
Belgium
{willems,pw}@montefiore.ulg.ac.be

Abstract

Partial-order methods make it possible to check properties of a concurrent system by state-space exploration without considering all interleavings of independent concurrent events. They have been applied to linear-time model checking, but so far only limited results are known about their applicability to branching-time model checking. In this paper, we introduce a general technique for lifting partial-order methods from linear-time to branching-time logics. This technique is shown to be applicable both to reductions that are applied to the structure representing the program before running the model checking procedure, as well as to reductions that can be obtained when model checking is done in an automata-theoretic framework. The latter are extended to branching-time logics by using the model-checking framework based on alternating automata introduced by Bernholtz, Vardi and Wolper.

1. Introduction

Model checking [6, 17, 20, 26] is by now a widespread technique for verifying concurrent programs. Its success is due to the fact that it is inherently implementable and is thus easily incorporated into tools. Its main drawback is that it requires the construction of the state space of the program being checked, state space which is often excessively large even for very small programs.

Fortunately several families of techniques have been introduced to cope with this "state-space explosion". One such family is that of "symbolic approaches" [5], which work by representing the state space implicitly, e.g. by a boolean formula coded as a Binary Decision Diagram (BDD) [3, 4], rather than explicitly. Another, on which we focus here, is that of partial-order approaches [21, 23, 24, 25, 12, 29, 13, 15, 18, 19]. These approaches make it possible to check meaningful properties of a state space while only considering a limited number of interleavings of independent concurrent actions, and hence avoiding much of the wastefulness of representing concurrency by interleaving. In practice they work by doing a state-space search in which only selected transitions are explored from each state. What varies between different methods of this family is the specific technique used for choosing the set of explored transitions.

Partial-order techniques were first introduced for checking simple properties such as absence of deadlock [21, 23, 12]. Later, they were extended to more elaborate properties, and in particular to model checking for linear-time temporal logic formulas [24, 25, 15, 18, 19]. Only recently has the problem of applying partial-order techniques to branching-time model checking been considered [10]. In that paper, it was shown that by being quite restrictive on the way the set of transitions to be explored is chosen, it was possible to preserve a subset of branching-time temporal logic properties (those not involving the "next-time" operator).

In this paper, we take an approach to partial-order methods for branching-time temporal logic inspired by [9]: *results for linear time can be lifted to branching time*. Indeed, branching-time temporal logic is obtained from linear-time temporal logic by prefixing formulas with path quantifiers and allowing atomic propositions to be arbitrary path quantified formulas (also called state formulas). Thus, as was already shown in [9], model checking a branching-time formula can be reduced to model checking a sequence of linear-time formulas. This is the technique we use. However, there is a slight complication in the context of partial-order techniques. Indeed, the arbitrary nesting allowed in branching time implies that independence with respect to the truth of atomic propositions has to be lifted to independence with respect to the truth of arbitrary state formulas, which is not automatic. By studying how this can be done, we get a very straightforward justification of the result presented in [10] as well as other results.

These other results are obtained in the context of the automata-theoretic approach to model checking [27]. In order to lift these results to branching-time temporal logic we need an automata-theoretic framework for branching-time model checking, which we take from [1]. There, it is shown that branching-time model checking algorithms can be cast into automata-theoretic terms by using alternating automata. Basically, the result that is established in [1] is that branching-time model checking can be reduced to the nonemptiness of an alternating 1-letter automaton. Furthermore, depending on the variant of branching-time temporal logic being considered, this automaton has some special structure that reflects the nesting of linear formulas within the branching formula, and this structure makes it possible to check its nonemptiness efficiently.

It is also this special structure that enables us to lift the results from linear-time model checking to branching time. Basically, we show that all partial-order techniques that can be used for automata-theoretic linear-time model checking can also be used for branching time, provided that state sub-formulas can be treated as atomic propositions. Of course, this proviso is nontrivial, but we give several easily implementable sufficient conditions under which it is satisfied. Finally, we discuss how these ideas can be incorporated into a model checking algorithm to obtain a partial-order version of local, or "on the fly", branching-time model-checking algorithms such as those in [28, 2].

The natural question is whether the approach we present here has significant practical value. We have not yet implemented it, but the way our algorithms are obtained makes it highly likely that the partial-order reduction they will achieve will be very similar to the one obtained when using their linear-time counterparts. Since this has been shown to be significant, we are quite optimistic.

2. Preliminaries

2.1. Temporal Logics

We consider the temporal logics LTL, CTL and CTL*, as well as their next-time-operator free versions LTL-$\bigcirc$, CTL-$\bigcirc$ and CTL*-$\bigcirc$. The logics LTL and LTL-$\bigcirc$ are linear-time temporal logics. Their formulas are built from a set of atomic propositions $\mathcal{P}$, Boolean connectives ($\wedge$, $\vee$, $\neg$), as well as the temporal operator U ("until") for LTL-$\bigcirc$, the operators U and $\bigcirc$ ("next time") for LTL.

The temporal logic CTL* distinguishes *state* and *path* formulas. State formulas are atomic propositions, path formulas preceded by one of the path quantifiers A ("for all paths") and E ("for some path"), or boolean combination thereof. Path formulas are LTL formulas in which the atomic propositions can be arbitrary state formulas. In CTL*-$\bigcirc$, the path formulas are restricted to be LTL-$\bigcirc$ formulas. In CTL, the path formulas are limited to a single occurrence of the operator U or of the operator $\bigcirc$. In CTL-$\bigcirc$, only the operator U is allowed.

We interpret temporal logic formulas over Kripke structures $K = (W, R, w^0, \mathcal{P}, L)$, where W is a set of states (worlds), $R \subseteq W \times W$ is a transition relation, w^0 is an initial state, $\mathcal{P}$ is a set of atomic propositions, and $L : W \rightarrow 2^{\mathcal{P}}$ maps each state to the set of atomic propositions true in that state. We suppose that the transition relation is total (i.e., for every $w \in W$ there exists $w' \in W$ such that $(w, w') \in R$). If R is not total, we can extend it by looping on the ending states or we can modify the corresponding semantic. A *path* in K is a sequence of states, $\pi = w_0, w_1, \ldots$ such that for every $i \geq 0$, $(w_i, w_{i+1}) \in R$. The notation π^j denotes the suffix of a path $\pi = w_0, w_1, \ldots$ starting with its state w_j. Linear-time formulas are interpreted over paths, branching-time state formulas over states of a Kripke structure.

The notation $K, w \models \varphi$ indicates that a state formula φ holds at the state w of the Kripke structure K. Similarly, $K, \pi \models \psi$ indicates that a path formula ψ holds at a path π of the Kripke structure K. The definition of the relation $\models$ is well known (see [8]). We only give the most noteworthy cases.

- $w \models A\psi$ iff for every path $\pi = w_0, w_1, \ldots$, with $w_0 = w$, we have $\pi \models \psi$.
- $w \models E\psi$ iff there exists a path $\pi = w_0, w_1, \ldots$, with $w_0 = w$, such that $\pi \models \psi$.
- $\pi \models \varphi$ for a state formula φ, iff $w_0 \models \varphi$ where $\pi = w_0, w_1, \ldots$
- $\pi \models \bigcirc\psi$ iff $\pi^1 \models \psi$.
- $\pi \models \psi_1 U \psi_2$ iff there exists $i \geq 0$ such that $\pi^i \models \psi_2$ and for all $0 \leq j < i$, we have $\pi^j \models \psi_1$.

2.2. From Temporal Logic to Automata

For linear-time, model-checking results are easily understood from the fact that temporal logic formulas can be translated to nondeterministic automata on infinite words [26, 11]. As is shown in [1], this approach can be extended to branching time by considering *alternating automata on infinite trees*.

First recall that an infinite tree is a set $T \subseteq \mathbb{N}^*$ such that if $x \cdot c \in T$ where $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$, then also $x \in T$, and for all $0 \leq c' < c$, $x \cdot c' \in T$. For a node $x \in T$, the number of nodes $xc \in T$ (successors of x) is the degree of x, which we will denote $d(x)$. A tree T such that for all $x \in T$, $d(x) \in \mathcal{D} \subseteq \mathbb{N}$ is called a $\mathcal{D}$-tree. We will consider trees where nodes are labeled with elements from an alphabet Σ, i.e. pairs (T, V) where $V : T \rightarrow \Sigma$ maps each node of T to a letter in Σ. A finite-state alternating tree automaton is a device for recognizing sets of trees. Precisely, it is a tuple $\mathcal{A} = (\Sigma, \mathcal{D}, Q, \delta, q_0, F)$ where

- Σ is the input alphabet (the alphabet of the labeled trees to be recognized),
- $\mathcal{D} \subset \mathbb{N}$ is a finite set of degrees (those allowed in the trees to be recognized) ,
- Q is a finite set of states,
- $\delta : Q \times \Sigma \times \mathcal{D} \rightarrow \mathcal{B}^+(\mathbb{N} \times Q)$, where $\mathcal{B}^+(X)$ denotes the positive boolean formulas built from the elements of the set X, and with the requirement that for every $k \in \mathcal{D}$, we have $\delta(q, \sigma, k) \in \mathcal{B}^+(\{0, \ldots, k-1\} \times Q)$, is the transition relation (intuitively, an element $(i, q) \in \mathbb{N} \times Q$ indicates that the state q is associated with the successor xi of the current node x), and
- $F = (G, B)$ with $G, B \subseteq Q$ is an acceptance condition.

The idea is that a tree is in the set defined by the automaton (is accepted by the automaton) if it is possible to associate states of the automaton to nodes of the tree in such a way that: the initial state is associated to the root of the tree, the transition relation is respected, and the acceptance condition is satisfied. To represent the way in which states of the automaton are associated with nodes of the tree, one introduces the notion of *run tree*. A run tree for an automaton $\mathcal{A}$ over a tree (T, V) is a tree (T_r, r) labeled by elements $(x, q) \in \mathbb{N}^* \times Q$ such that $x \in T$. So, each node of T_r corresponds to a node of T, but there can be several nodes of T_r associated to a node of T. Now, a run tree T_r is an accepting run tree for T if the following conditions are satisfied.

- The initial state is associated with the root of T: $r(\varepsilon) = (\varepsilon, q_0)$.
- The transition relation is respected: for all y in T_r, if $r(y) = (x, q)$ then there is a possibly empty set $S = \{(c_0, q_0), (c_1, q_1), \ldots, (c_n, q_n)\} \subseteq \{0, \ldots, d(x)-1\} \times Q$ that makes $\delta(q, V(x), d(x))$ true, and such that for all $0 \leq i \leq n$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = (x \cdot c_i, q_i)$.
- The acceptance condition is satisfied: for every infinite path in T_r starting at the root, there are infinitely many y such that $r(y) = (x, q)$ with $q \in G$ or only finitely many y such that $r(y) = (x, q)$ with $q \in B$[1].

The results giving the translation from temporal logics to automata are the following (see [16]).

[1]The acceptance condition we use can be viewed as a very special type of Rabin acceptance condition. It is tuned towards expressing the translation of CTL* formulas.

Theorem 2.1 *Given a* CTL *formula φ and a set $\mathcal{D} \subset \mathbb{N}$, we can construct an alternating tree automaton $\mathcal{A}_{\mathcal{D},\varphi}$ of size $O(|\mathcal{D}| * |\varphi|)$ such that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\varphi})$ is exactly the set of tree models, with degree in $\mathcal{D}$, satisfying φ.*

Theorem 2.2 *Given a* CTL* *formula φ and a set $\mathcal{D} \subset \mathbb{N}$, we can construct an alternating tree automaton $\mathcal{A}_{\mathcal{D},\varphi}$ of size $O(|\mathcal{D}| * 2^{|\varphi|})$ such that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\varphi})$ is exactly the set of $\mathcal{D}$-trees satisfying φ.*

Furthermore, the automata obtained from CTL and CTL* formulas have a very special structure. Their set of states Q can be partitioned into a finite partially ordered set of components Q_i. The order on these components is such that transitions from a state in a given component only involve states in the same or lower components. In addition these components are all in one of three categories:

1. *Transient*, the transitions from states in a transient component only involve states in lower components,
2. *Existential*, the component is not transient and all occurrences of states from the component in a transition from one of its states are *disjunctively* related,
3. *Universal*, the component is not transient and all occurrences of states from the component in a transition from one of its states are *conjunctively* related.

The acceptance condition $F = (G, B)$ of the automaton also has some properties related to this partitioning: for existential components Q_e, $Q_e \cap B = \emptyset$ and for universal components Q_u, $Q_u \cap G = \emptyset$.

This structure of the automaton is in fact directly linked to the structure of the formula: existential components correspond to $E\psi$ formulas, universal components to $A\psi$ formulas and transient components to other state formulas. Also notice that for a formula $E\psi$ where ψ is a linear formula only built from atomic propositions, the automaton has a single existential component and hence is simply a nondeterministic automaton.

2.3. Model Checking Algorithms

To check that a Kripke structure K satisfies a CTL or CTL* formula φ, one simply constructs the automaton $\mathcal{A}_{\mathcal{D},\varphi}$ where $\mathcal{D}$ is the set of degrees of K, computes the product $K \times \mathcal{A}_{\mathcal{D},\varphi}$, and checks that it is nonempty. As is shown in [1, 16], this yields optimal algorithms for the model-checking problem.

Also, notice that the product $K \times \mathcal{A}_{\mathcal{D},\varphi}$ can be viewed as an input-less alternating word automaton, i.e. as a Boolean transition system, or Boolean graph. Precisely, it is a structure $A_{K,\varphi} = (Q, \delta, q_0, F)$ where Q is a set of states, $\delta : Q \rightarrow \mathcal{B}^+(Q)$ a Boolean transition relation, and

$F = (G, B)$ with $G, B \subseteq Q$ an acceptance condition. Furthermore, this boolean transition system inherits the structure of existential, universal, and transient components from $\mathcal{A}_{\mathcal{D},\varphi}$. It is nonempty if its *evaluation* assigns the truth value $\mathbf{true}$ to q_0. The evaluation of a Boolean transition system is an assignment of truth values to the elements of Q such that for each state q, the truth value of q is identical to the one given by $\delta(q)$; a state s within an existential component whose labeling is not forced by δ is labeled by $\mathbf{true}$ iff it is possible to reach from s a state in G which is also reachable from itself; and, dually, a state s within a universal component whose labeling is not forced by δ is labeled by $\mathbf{false}$ iff it is possible to reach from s a state in B which is also reachable from itself.

3. Partial-Order Methods

Partial-order methods take into account the fact that the Kripke structure $K = (W, R, w^0, \mathcal{P}, L)$ to be model checked is obtained from a concurrent program. This implies that the moves possible in the Kripke structure (pairs $(w, w') \in R$) can be grouped into equivalence classes called *transitions*. The intuition is that moves in the same transition correspond to the same "concurrent program transition", i.e. the same processes executing the same actions. Furthermore, one can define an *independence relation* on transitions: two transitions are independent if they cannot enable or disable each other, and if they lead to the same state whatever order they are executed in.

To formalize this, we add to the definition of a Kripke structure $K = (W, R, w^0, \mathcal{P}, L)$ a set $\mathcal{T}$ of transitions that partitions the moves $(w, w') \in R$ (with the requirement that moves from the same state all belong to different transitions) and an independence relation $\mathcal{I} \subseteq \mathcal{T} \times \mathcal{T}$ that must satisfy the following definition.

Definition 3.1 *A relation $\mathcal{I} \subseteq \mathcal{T} \times \mathcal{T}$ is an* independence *relation for a Kripke structure K iff for all $(t_1, t_2) \in \mathcal{I}$ the following conditions are true in all states w of K:*

1. *if t_1 (t_2) is enabled in w, i.e. there is a move $(w, w') \in t_1$, and*[2] *$w \xrightarrow{t_1} w'$ ($w \xrightarrow{t_2} w'$), then t_2 (t_1) is enabled in w iff t_2 (t_1) is enabled in w' (independent transitions can neither disable nor enable each other); and*
2. *if t_1 and t_2 are enabled in w, then there is a unique state w' such that both $w \overset{t_1t_2}{\Rightarrow} w'$ and $w \overset{t_2t_1}{\Rightarrow} w'$ (commutativity of enabled independent transitions).*

In practice, one of course needs easily checkable sufficient conditions for independence. See [13, 29] for examples of such conditions. The way independence is exploited in partial-order methods is to use it as a guide for not exploring all moves from a state of the Kripke structure. Precisely, one can check properties of the Kripke structure by doing a depth-first search through the structure that only considers a subset of the possible moves from each state, a *selective search* in the terminology of [13]. The idea is that one can choose to ignore some moves that are independent with respect to those that are chosen. Concretely, one needs a slightly stronger condition which is embodied by the notion of *persistent set* [14, 13].

Definition 3.2 *A set T of transitions enabled in a state w is* persistent *in w if and only if, for all nonempty sequences of transitions*

$$w = w_0 \xrightarrow{t_0} w_1 \xrightarrow{t_1} w_2 \ldots \xrightarrow{t_{n-1}} w_n \xrightarrow{t_n} w_{n+1}$$

from w in K, and including only transitions $t_i \notin T$, t_n is independent with respect to all transitions in T.

Notice that the set of all enabled transitions in a state is trivially persistent. Practical techniques for computing persistent sets can be found in [13, 29]. Similarly, the *stubborn sets* of Valmari [22] are persistent sets obtained by a particular algorithm.

A first result is that, if one builds a subset K_R of K built by doing a selective search that selects in each state a nonempty persistent set of transitions, all deadlock states of K (assuming then, of course, that the transition relation of K is not total) also appear in K_R. One can also go beyond deadlock detection. Define a *local property* to be one that cannot be changed by two distinct independent transitions (in practice local properties are those that only concern the state of a single process or of a single shared data object). Then, the reachability of a state satisfying a local property is preserved in any subset K_R of K obtained by a selective search that selects in each explored state w a nonempty set of transitions such that

(persistent) the set is persistent and nonempty,

(cycle-proviso) there is no cycle $w = w_0 \xrightarrow{t_0} w_1 \xrightarrow{t_1} w_2 \ldots \xrightarrow{t_{n-1}} w_n \xrightarrow{t_n} w$ such that for all w_i, $0 \leq i \leq n$, the set of selected transitions is different from the set of all enabled transitions.

Again there are simple practical means for ensuring the second condition (see for instance [13]). Finally, note that one can further reduce the set of selected transitions by using the *Sleep Set* technique of [12, 13] and still preserve the reachability of states satisfying local properties.

[2] We write $w \xrightarrow{t} w'$ to mean that the transition t leads from the state w to the state w' and $w \overset{u}{\Rightarrow} w'$ to mean that the sequence of transitions u leads from w to w'.

4. Partial-Order Methods for Linear-Time Model Checking

The next natural question is whether the partial-order techniques we have described can be used in the context of model-checking. For linear-time logics the problem has been well studied. There are two main approaches. The first attempts to reduce the Kripke structure before applying the model-checking procedure. The second applies the partial-order technique during a direct computation of the product $K \times A_\varphi$ of the Kripke structure and the automaton for the formula (in practice, for linear time, the negation of the formula that needs to be checked). We briefly review the known results.

4.1. Pre-Reduction Techniques

The idea is to make sure the formula will be insensitive to the selection of transitions made by the selective search. One starts by noticing that formulas with next ($\bigcirc$) operators are very sensitive to transition selection. So, the first step is only to consider LTL-$\bigcirc$. The second step is to notice that two transitions that have moves between states in which atomic propositions appearing in the formula have different values cannot safely be considered as independent. This leads to the introduction of the following definition.

Definition 4.1 *For a given Kripke structure K and a set of atomic propositions $\mathcal{P}$, a transition t is said to be $\mathcal{P}$-*invisible *in a state w if for any move $(w, w') \in t$, $L(w)$ and $L(w')$ coincide on $\mathcal{P}$.*

It might appear that when selecting transitions in a state, it would be safe to add to the previous requirements the one that, either one selects only transitions that are invisible with respect to the atomic propositions of the formula being checked, or one selects all visible transitions. There are two problems with this. The first is that selecting visible transitions (and omitting invisible ones) might lead to ignoring an infinite path on which nothing changes, but that does satisfy the formula. The second is that invisibility in the state in which the transitions are selected is not sufficient: one also needs invisibility in some of the states reachable from there. To solve the first problem one requires that it is always the selected transitions that are invisible. To solve the second, we introduce the following definition.

Definition 4.2 *Given a state w of a Kripke structure K, a set of transitions T, which is persistent in w, is $\mathcal{P}$-*persistently-invisible *in w, if for all nonempty sequences of transitions*

$$w = w_0 \xrightarrow{t_0} w_1 \xrightarrow{t_1} w_2 \ldots \xrightarrow{t_{n-1}} w_n$$

from w in K, and including only transitions $t_i \notin T$, all elements of T are $\mathcal{P}$-invisible in w_n.

This might appear as a rather complicated condition. In practice one rather checks for $\mathcal{P}$-invisibility in all states, but the weaker notion of Definition 4.2 will enable us to lift results from linear-time to branching time. The notion of $\mathcal{P}$-persistently-invisible is incorporated into the transition selection procedure by the following criterion.

$\mathcal{P}$-persistently-invisible If the set of selected transitions is not the set of all enabled transitions, it is $\mathcal{P}$-persistently-invisible.

With this additional restriction, it can be shown that the same LTL-$\bigcirc$ formulas are true in all states that appear both in K_R and K (we say that K_R *preserves* LTL-$\bigcirc$ formulas). The theorem is the following [22, 18].

Theorem 4.1 *A reduced Kripke structure K_R constructed with a selective search in which the selected sets of transitions satisfy the conditions* **persistent**, **cycle-proviso**, *and* **$\mathcal{P}$-persistently-invisible** *preserves all* LTL-$\bigcirc$ *formulas built from $\mathcal{P}$.*

4.2. On the Fly Reduction Techniques

In this approach, one does not do a selective search on the Kripke structure, but on the product of the Kripke structure and of the automaton for the formula. In the case of linear-time, this product is simply a disjunctive graph whose states are composed of a state of the Kripke structure and a state of the automaton. Fundamentally, all one needs to do is to define the notion of independence between transitions directly for the product graph rather than only for the Kripke structure. This is done as follows. First, the partition of moves into transitions defined on the Kripke structure is refined to take into account the moves of the automaton: two moves in the product structure are only in the same transition if they are in the same Kripke transition and correspond either to the same move or to no move of the formula automaton. Independence is then still defined exactly as in Definition 3.1 except that it is now relative to the product graph, with the consequence that whether a transition interacts or not with the formula is directly taken into account by the independence relation. Again various sufficient conditions can be obtained for guaranteeing independence.

As is shown in [15], a partial-order technique that preserves reachability of local properties also preserves repeated nontrivial reachability of local properties. So, one might believe that it is sufficient to impose conditions **persistent** and **cycle-proviso** on the selected sets of transitions. This is not so, since one might loose an infinite accepting computation in which the automaton for the formula does not move. In [15] this problem is solved by ensuring that no such computations exist. Another solution [25] is to notice that such infinite immobile computations must stay in a G state of the automaton (remember that we are dealing with an existential automaton whose acceptance condition reduces to going infinitely often through the G part of the acceptance condition). So the rule we apply is, when the formula automaton is in a G state, never to exclusively select transitions that interact with it, i.e. that can disable one of its moves or change its state. This rule is embodied in the following condition on selected sets of transitions:

stay-accept If not all transitions are selected and if the formula automaton is in an accepting G state, then the set of selected transitions must contain transitions that do not interact with the formula automaton, i.e. that cannot disable one of its moves or change its state.

The result is then the following

Theorem 4.2 *A selective search through the structure $K \times A_\psi$ preserves nonemptiness if the selected sets of transitions satisfy (with respect to independence in $K \times A_\psi$) the conditions* **persistent**, **cycle-proviso**, *and* **stay-accept**.

It is interesting to note that this approach is not limited to LTL-$\bigcirc$. However, formulas with next operators do induce many dependencies and hence drastically limit the improvement that can be obtained with the partial-order techniques.

Finally, let us note that Theorem 4.2 still holds when one restricts the set of selected transitions using the Sleep Set technique of [12, 13], but with an additional condition: the sleep set of G states must always be forced to be the empty set. Indeed, sleep sets preserve reachability from states with an empty sleep set. Since the initial state always has an empty sleep set, reachability from the initial state is preserved. However, here we also need to preserve reachability from G states. Thus, we need to impose that the sleep set of these states always be taken to be empty. Finally, notice that implementing the test of nonemptiness of the reduced structure $K \times A_\psi$ in a purely on-the-fly way would require adapting the technique described in [7].

5. Partial-Order Methods for Branching-Time Model Checking

5.1. Pre-Reduction Techniques

As pointed out in [9], branching-time model checking can be reduced to a sequence of linear-time model checking problems. One might thus expect that a partial-order construction that preserves linear-time formulas also preserves branching-time formulas. Does this apply to Theorem 4.1, i.e does the same theorem also hold for CTL*-$\bigcirc$? Not quite! Indeed, Theorem 4.1 relies on the fact that, when not all transitions are selected, the selected ones preserve the truth of the atomic propositions from which the LTL-$\bigcirc$ formulas are built. In CTL*-$\bigcirc$, the atomic propositions appearing in LTL-$\bigcirc$ path formulas are arbitrary CTL*-$\bigcirc$ state formulas. What we need is to replace the notion of invisibility by the following notion of invisibility with respect to the formulas of a logic.

Definition 5.1 *For a given Kripke structure K, and a set of formulas $\mathcal{L}(\mathcal{P})$ built from a set of atomic propositions $\mathcal{P}$, a transition t is said to be $\mathcal{L}(\mathcal{P})$-invisible in a state w if for any move $(w, w') \in t$, w and w' satisfy the same formulas of $\mathcal{L}(\mathcal{P})$.*

Then, $\mathcal{L}(\mathcal{P})$-persistently-invisible is defined similarly to what is done in Definition 4.2. Now, if in Theorem 4.1 we replace the condition $\mathcal{P}$**-persistently-invisible** by a similar CTL*-$\bigcirc(\mathcal{P})$**-persistently-invisible** then it will also hold for CTL*-$\bigcirc$. We can also state a more precise version of this by considering the nesting depth of path quantifiers in CTL*-$\bigcirc$ formulas. Define CTL$^*_k$-$\bigcirc(\mathcal{P})$ to be the set of CTL*-$\bigcirc(\mathcal{P})$ formulas with at most k levels of nesting of path quantifiers. We then have the following which is proved exactly as Theorem 4.1.

Lemma 5.1 *Consider a Kripke structure K and a reduced structure K_R obtained by a selective search. If K_R preserves all* CTL$^*_k$-$\bigcirc(\mathcal{P})$ *formulas, and if K_R is constructed with a selection of transitions that satisfies the conditions* **persistent**, **cycle-proviso**, *and* CTL$^*_k$-$\bigcirc(\mathcal{P})$**-persistently-invisible**, *then K_R preserves all* CTL$^*_{k+1}$-$\bigcirc(\mathcal{P})$ *formulas.*

Of course, what one needs is an easy criterion for guaranteeing CTL$^*_k$-$\bigcirc(\mathcal{P})$-invisibility for the relevant k. In [10], adding the following condition to $\mathcal{P}$-persistent-invisibility is shown to do the job for any k:

singleton The set of selected transitions is either a singleton or the set of all enabled transitions.

That this works is a consequence of the following Lemma.

Lemma 5.2 *Consider a Kripke structure K and a reduced structure K_R obtained by a selective search. If K_R preserves all* CTL$^*_k$-$\bigcirc(\mathcal{P})$ *formulas, and if K_R is constructed with a selection of transitions that satisfies the conditions* **persistent**, **cycle-proviso**, CTL$^*_k$-$\bigcirc(\mathcal{P})$**-persistently-invisible**, *and* **singleton**, *then the set of transitions selected in K_R also satisfy the condition* CTL$^*_{k+1}$-$\bigcirc(\mathcal{P})$**-persistently-invisible**.

Proof Sketch. Indeed, consider a move (w, w') of a transition that is a singleton set satisfying all conditions in the Lemma statement. By Lemma 5.1 the Kripke structure K^0_R obtained by only selecting the move (w, w') of t in w preserves all CTL$^*_{k+1}$-$\bigcirc(\mathcal{P})$ formulas. Also, notice that, by the definition of persistent sets and by the definition of CTL$^*_k$-$\bigcirc(\mathcal{P})$-persistently-invisible, the same applies to any structure K_R^i obtained by only selecting a move (w_i, w_i') of t in a state w_i accessible from w by transitions different from t. So, to show that the transition t is CTL$^*_{k+1}$-$\bigcirc(\mathcal{P})$-persistently-invisible in w we only need to show that it is CTL$^*_{k+1}$-$\bigcirc(\mathcal{P})$-invisible in the state w_i of each of the structures K_R^i.

This however is an immediate consequence of the fact that in K_R^i, the move (w_i, w_i') is the only one from w_i, that we are dealing with $\bigcirc$-free formulas and that we have CTL$^*_k$-$\bigcirc(\mathcal{P})$-invisibility in w_i. □

Using Lemmas 5.1 and 5.2 in an induction on the nesting depth of path quantifiers in CTL*-$\bigcirc$, one then directly obtains the result established in [10].

Theorem 5.3 *A reduced Kripke structure K_R constructed with a selective search in which the selected sets of transitions satisfy the conditions* **persistent**, **cycle-proviso**, $\mathcal{P}$**-persistently-invisible**, *and* **singleton** *preserves all* CTL*-$\bigcirc(\mathcal{P})$ *formulas.*

Notice however that in [10] the condition that is used is not $\mathcal{P}$**-persistently-invisible**, but the stronger $\mathcal{P}$-invisible in all states. Refining the analysis of what was precisely needed has both enabled us to lift the linear-time result to branching-time and to get a stronger result. Furthermore, the proof technique we have used also opens up the possibility of lifting other partial-order techniques from linear-time to branching-time.

5.2 On the Fly Reduction Techniques

Let us now try to apply partial-order methods directly to the construction of the Boolean graph representing the product of the Kripke structure and of the automaton for the formula being checked. A first observation is that partial-order reductions are only obtainable when transitions of the Kripke structure are explored. This happens in the existential and universal components of the Boolean graph, i.e. in the components that correspond to checking $E\psi$ or $A\psi$ formulas. Consider first an $E\psi$ component Q_e. The Boolean transition bt from a state $q \in Q_e$ only involves disjunctively related states of Q_e. This means that, if we have already evaluated the states of lower components, which is the case when actually searching through the Boolean graph, and substitute the truth values that are obtained in bt, we obtain a disjunction D_q of states of Q_e. Furthermore, the moves from q to the states of D_q do correspond to moves within the Kripke structure. So, it seems quite natural to select among them with the help of partial-order techniques. Our strategy is to use the results of Section 4.2 within the component Q_e. For this, we consider Q_e with its transitions simplified given the values obtained for states in lower components. Precisely, we use the following notion.

Definition 5.2 *Consider an existential component Q_e of a Boolean graph $A_{K,\varphi} = (Q, \delta, q_0, (G, B))$ obtained as the product of a Kripke structure and the automaton for a formula. The* simplified *Q_e is the disjunctive graph $(Q_e, \delta_e, G \cap Q_e)$ where, for $q \in Q_e$, $\delta_e(q)$ is $\delta(q)$ where states in components lower than Q_e have been assigned the value given by the evaluation of the Boolean graph.*

A simplified existential component is then exactly the type of structure we considered in Section 4.2 and Theorem 4.2 can be used. The only practical problem is that, to apply Theorem 4.2, we need to compute persistent sets with respect to independence defined on the simplified existential component. How can this be done? Of course, we cannot explore that whole simplified component, this would defeat the purpose the partial-order method. We need sufficient conditions.

Transitions in the simplified existential component correspond to a transition in the Kripke structure and to a move between a given pair of states of the automaton. For two transitions t_1 and t_2 to be independent, we first require that the corresponding transitions in the Kripke structure be independent. The next step is to require that they are independent with respect to the automaton for the formula. One way to this is to distinguish transitions that do not interact with the formula automaton and those that do. Two transitions are then independent if at most one interacts with the automaton. We are then left with finding sufficient criteria for transitions "not to interact" with the formula automaton. We use two requirements. The first is that the transition does not modify the state of the formula automaton. The second is that it does not modify the choices of the formula automaton. The first requirement is easy to check. For the second, a sufficient condition is that the truth values obtained from lower components in order to simplify the Boolean transition function to its disjunctive form within the component be unchanged by the transition. There are three conditions under which this can be guaranteed :

- the Boolean transition simplifies trivially without values from lower components or
- the Boolean transition only involves elements of lower components corresponding to atomic formulas, and the Kripke transition does not modify these (this is similar to $\mathcal{P}$-invisibility) or
- one adds the **singleton** requirement to the partial-order selection method which makes it possible to lift invisibility of atomic propositions to invisibility of arbitrary formulas as is done in Lemma 5.2.

We now have to consider the case of universal components. The treatment of these is perfectly dual to the one of existential components.

The results we have given can be incorporated into a partial-order on the fly model-checking algorithm which operates by searching through the Boolean graph obtained from the Kripke structure and the alternating automaton for the formula and applying the partial-order reductions described above in its existential and universal components. This could be viewed as a partial-order extension of the algorithm given in [2]. Notice that the partial-order reductions we use are in fact the ones for linear-time logics and that we can take advantage of all results known in this context. For instance, the Sleep Set Technique of Godefroid [12, 13] could probably also be adapted to our context.

Let us finally illustrate the reductions that can ve obtained with a simple example. Consider the following CTL formula: $\varphi = \forall\Box(\varphi_1 \vee \varphi_2)$, where $\varphi_1 = \forall\Diamond q$ and $\varphi_2 = \forall\Diamond\neg p$.

The transition relation of the corresponding weak alternating automaton $A_{\mathcal{D},\varphi}$ defined in [1] is given in the tables below ($k \neq 0$):

s	$\delta(s, \{p\}, k)$
φ	$\wedge_{c=0}^{k-1}(c,\varphi) \wedge (\wedge_{c=0}^{k-1}(c,\varphi_1) \vee \wedge_{c=0}^{k-1}(c,\varphi_2))$
φ_1	$\wedge_{c=0}^{k-1}(c,\varphi_1)$
φ_2	$\wedge_{c=0}^{k-1}(c,\varphi_2)$
$\neg p$	**false**
q	**false**

s	$\delta(s, \{p,q\}, k)$	$\delta(s, \{q\}, k)$	$\delta(s, \emptyset, k)$
φ	$\wedge_{c=0}^{k-1}(c,\varphi)$	$\wedge_{c=0}^{k-1}(c,\varphi)$	$\wedge_{c=0}^{k-1}(c,\varphi)$
φ_1	**true**	**true**	$\wedge_{c=0}^{k-1}(c,\varphi_1)$
φ_2	$\wedge_{c=0}^{k-1}(c,\varphi_2)$	**true**	**true**
$\neg p$	**false**	**true**	**true**
q	**true**	**true**	**false**

Consider now the Kripke structure of figure 1 (the numbers besides the nodes are just labels). This Kripke structure can not be simplified with the partial-order method of [10].

The Boolean graph obtained as the product of the alternating automaton for the formula φ and of the Kripke structure above is given in Figure 2. It is a "weak Boolean graph" in the sense that it inherits the structure of the weak alternating automaton constructed for φ. It has three components : $E_1 \leq E_3$ and $E_2 \leq E_3$. The set of accepted nodes is E_3. The truth value of the node $(1,\varphi)$ is **false** which means that φ is not satisfied by the Kripke structure.

Our reduction technique allows us to only explore the transition labeled a from the state $(1,\varphi)$. Indeed, the transition labeled a is a persistent set since it is independent with respect to the transitions labeled b and c. Indeed, these transitions are independent in the Kripke structure and the latter two do not interact with the automaton for the formula.

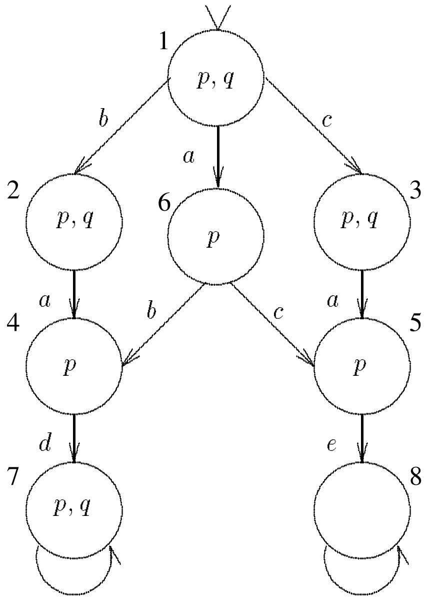

Figure 1. Kripke structure

6. Conclusions

The contribution of this paper is to show that partial-order methods for branching-time model checking can be obtained in a fairly systematic way from similar methods for linear-time. The main idea is that results can be lifted from linear-time to branching-time exactly as was initially shown in [9]. This gives a very simple proof of an earlier result on the subject, leads to interesting insights, and yields new methods based on the automata-theoretic approach to model checking. Of course, there is a certain unavoidable technical complexity in the subject, but we believe that our approach minimizes it.

Concerning the practical effectiveness of our automata-theoretic algorithm, we have not yet implemented it and obtained practical results. However, there are reasons to believe that it will give similar reductions to those that can be obtained with the methods for linear-time logic that it extends.

As far as further work, one challenging question we see in this area is whether partial-order model-checking can be extended to fragments of the μ-calculus that, unlike CTL and CTL* cannot be reduced to nested boolean combinations of linear formulas.

References

[1] O. Bernholtz, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In

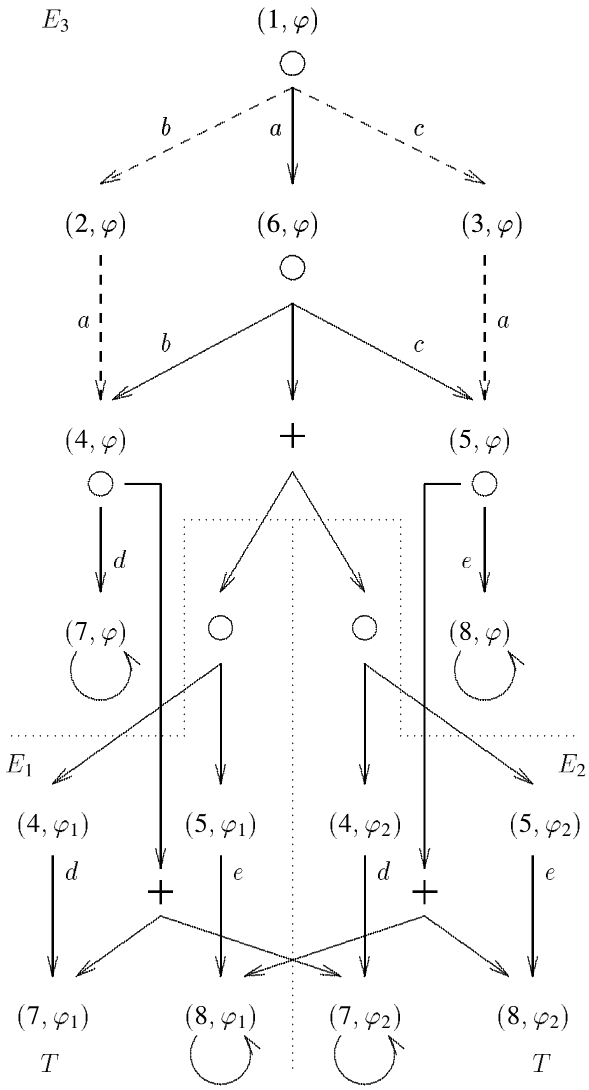

Figure 2. Weak boolean graph

Computer Aided Verification, Proc. 6th Int. Workshop, Stanford, California, June 1994. Lecture Notes in Computer Science, Springer-Verlag. full version available from authors.

[2] G. Bhat, R. Cleaveland, and O. Grumberg. Efficient on-the-fly model checking for ctl*. In *Proceedings of the Tenth Symposium on Logic in Computer Science*, San Diego, June 1995. To appear.

[3] R. E. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[4] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[5] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[6] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[7] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[8] E. Emerson. Temporal and modal logic. *Handbook of theoretical computer science*, pages 997–1072, 1990.

[9] E. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages*, pages 84–96, New Orleans, January 1985.

[10] R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. In *Proceedings of the 3d Israel Symposium on the Theory of Computing and Systems (ISTCS)*, pages 130–140. IEEE Computer Society Press, 1995. Submitted for journal publication.

[11] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. 15th Work. Protocol Specification, Testing, and Verification*, Warsaw, June 1995. North-Holland.

[12] P. Godefroid. Using partial orders to improve automatic verification methods. In *Proc. 2nd Workshop on Computer Aided Verification*, volume 531 of *Lecture Notes in Computer Science*, pages 176–185, Rutgers, June 1990. Springer-Verlag.

[13] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems*. PhD thesis, University of Liège, Liège, Belgium, November 1994.

[14] P. Godefroid and D. Pirottin. Refining dependencies improves partial-order verification methods. In *Proc. 5th Conference on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 438–449, Elounda, June 1993. Springer-Verlag.

[15] P. Godefroid and P. Wolper. A partial approach to model checking. *Information and Computation*, 110(2):305–326, May 1994.

[16] O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. full version available from authors, 1995.

[17] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages*, pages 97–107, New Orleans, January 1985.

[18] D. Peled. All from one, one for all: on model checking using representatives. In *Proc. 5th Conference on Computer Aided Verification*, Elounda, June 1993. Lecture Notes in Computer Science, Springer-Verlag.

[19] D. Peled. Combining partial order reductions with on-the-fly model-checking. In *Computer Aided Verification, Proc. 6th Int. Workshop*, Stanford, California, June 1994. Lecture Notes in Computer Science, Springer-Verlag.

[20] J. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th Int'l Symp. on Programming*, volume 137, pages 337–351. Springer-Verlag, Lecture Notes in Computer Science, 1981.

[21] A. Valmari. Error detection by reduced reachability graph detection. In *Proc. 9th European Workshop on Application and Theory of Petri Nets*, pages 95–112, Venice, 1988.

[22] A. Valmari. A stubborn attack on state explosion. In *Proc. 2nd Workshop on Computer Aided Verification*, volume 531 of *Lecture Notes in Computer Science*, pages 156–165, Rutgers, June 1990. Springer-Verlag.

[23] A. Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515, Bonn, June 1991. Springer-Verlag.

[24] A. Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1:297–322, 1992.

[25] A. Valmari. On-the-fly verification with stubborn sets. In *Proc. 5th Conference on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 397–408, Elounda, June 1993. Springer-Verlag.

[26] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.

[27] M. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–21, April 1986.

[28] B. Vergauwen and J. Lewi. A linear local model checking algorithm for CTL. In *Proc. CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 447–461, Hildesheim, August 1993. Springer-Verlag.

[29] P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In *Proc. CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 233–246, Hildesheim, August 1993. Springer-Verlag.