# Complexity of Compositional Model Checking of Computation Tree Logic on Simple Structures

Krishnendu Chatterjee[1], Pallab Dasgupta[2⋆], and P.P. Chakrabarti[2*]

[1] EECS, University of California, Berkeley,
[2] Dept. of Computer Science & Engg., Indian Institute of Technology, Kharagpur
c_krish@cs.berkeley.edu, {pallab,ppchak}@cse.iitkgp.ernet.in

**Abstract.** Temporal Logic Model Checking is one of the most potent tools for the verification of finite state systems. Computation Tree Logic (CTL) has gained popularity because unlike most other logics, CTL model checking of a single transition system can be achieved in polynomial time. However, in most real-life problems, specially in distributed and parallel systems, the system consist of a set of concurrent processes and the verification problem translates to model check the composition of the component processes. Since explicit composition leads to state explosion, verifying the system without actually composing the components is attractive, even for possibly restrictive class of systems. We show that the problem of compositional CTL model checking is PSPACE complete for the class of systems composed of components that are tree-like transition structure and do not interact among themselves. For the simplest forms of existential and universal CTL formulas model checking turns out to be NP complete and coNP complete, respectively. The results hold for both synchronous and asynchronous composition.

## 1 Introduction

Temporal logic model checking [2, 7] has emerged as one of the most powerful techniques for verifying temporal properties of finite-state systems. The correctness property of the system that needs to be verified is specified in terms of a temporal logic formula. Model checking has been extensively studied for two broad categories of temporal logics, namely *linear time temporal logic* (LTL) and *branching time temporal logic* [3]. The branching time temporal logic, *Computation Tree Logic* (CTL) [2], is one of the most popular temporal logics in practice. CTL allows us to express a wide variety of branching time properties which can be verified in polynomial time (that is, the time complexity of CTL model checking is polynomial in the size of the state transition system times the length

⋆ Pallab Dasgupta and P.P.Chakrabarti thank the Dept. of Science & Tech., Govt. of India, for partial support of this work.

A. Sen et al. (Eds.): IWDC 2004, LNCS 3326, pp. 98–109, 2004.

of the CTL formula). CTL is also a syntactically elegant, expressive logic which makes CTL model checking computationally attractive as compared to the other logics like LTL and CTL* which are known to be PSPACE complete [7].

Given an explicit representation of a state transition system, $M$, CTL model checking is polynomial in the size of $M$ times the length of the formula. In practice systems are seldom represented explicitly as a single transition structure. Generally a large system consists of several concurrent components that run parallely or in a distributed environment. Hence for verification of parallel and distributed systems it is important to be able to verify systems that are described as a set of concurrent processes. Given a set of concurrent components the complete transition system can be a synchronous [1], [7] or asynchronous composition of the components [7]. The composition of the components into a single transition structure is accompanied by the state-explosion problem as the size of the complete state transition structure will be the product of the size of the component transition structures. Therefore the ability to perform model checking without explicit composition is an attractive proposition, even for possibly restrictive class of systems. There have been several approaches to this sort of compositional model checking [7].

Model checking of logics like LTL and CTL* are known to be PSPACE complete for a state transition system. So the complexity of compositional model checking for such logics will be computationally hard as well. CTL model checking is known to be polynomial for a state transition system [2]. Given a set of $k$ concurrent transition systems of size $|S|$, where $k$ is a constant, CTL model checking on the global system which is a composition of the component systems can be achieved in time polynomial in $|S|^k$ times the length of the formula. This is done by composing the components into a single system (of the size $O(|S|^k)$) and applying the *CTL model checking algorithm* on it. If $k$ is not an constant this approach of model checking does not produce a polynomial time solution.

In this paper we study complexity of model checking of CTL properties of a set of concurrent processes considering several modes of composition, namely synchronous and asynchronous composition. We show that the problem is hard even for a very restrictive class of concurrent systems. We consider system composed of components that tree-like transition systems, i.e., the components are trees with leaves having self-loops. Moreover, the components do not communicate among themselves and all these components are specified as an explicit representation of the system. We prove that the problem of CTL model checking is PSPACE complete. However, a PSPACE-upper bound can be proved for a more general class of concurrent systems. We also show that the problem of checking simple *existential* CTL formulas like $E(B\ U\ B)$ and *universal* formulas like $A(B\ U\ B)$, where $B$ is a Boolean formula, is NP complete and coNP complete, respectively. We also show that the problem of reachability of two states of such tree-like structures can be answered in time linear in the size of the input. All the results hold for both synchronous and asynchronous composition. Our result proves that the compositional model checking for CTL is hard for very restrictive classes of systems and the problem is inherently computationally hard.

This paper is organized as follows. In Section 2 we define *tree-like* kripke structures and the synchronous, asynchronous composition of a set of tree-like kripke structures; and also describe the syntax and semantics of CTL in Section 2. In Section 3 we study the complexity of model checking of CTL on a system which is the composition of a set of tree-like kripke structures. In Section 4 we analyze the complexity of reachability of two states.

## 2 Tree-like Kripke Structure

We formally define a tree-like kripke structure and the composition of a set of tree-like kripke structures below.

**Definition 1 (Tree-like Kripke Structure).** *A* tree-like kripke structure, *$T_i = \langle S_i, s_{0i}, \mathcal{R}_i, \mathcal{L}_i, \mathcal{AP}_i \rangle$, consists of the following components:*

- *$S_i$ : finite set of states and $s_{0i} \in S_i$ is the initial state.*
- *$\mathcal{AP}_i$ is the finite set of atomic propositions.*
- *$\mathcal{L}_i : S_i \rightarrow 2^{\mathcal{AP}_i}$ — labels each state $s \in S_i$ with a set of atomic propositions true in $s$.*
- *$\mathcal{R}_i \subseteq S_i \times S_i$ is the transition relation with the restriction that the transition relation graph is a tree with leaves having self-loops. The transition relation is also total, i.e., for every state $s_i \in S_i, \exists\ s'_i \in S_i$ such that $\mathcal{R}_i(s_i, s'_i)$.* ∎

**Definition 2 (Composition).** *Let $T = \{T_1, T_2, \ldots, T_m\}$ be a set of $m$ tree-like kripke structures. The synchronous, asynchronous and strict asynchronous composition of the tree-like kripke structures in $T$ is denoted by $T_S, T_A$ and $T_{SA}$, respectively. The set of states, initial state, the set of atomic proposition and the labeling function is same for $T_S, T_A$ and $T_{SA}$ and is defined as follows: 1. $S = S_1 \times S_2 \times ... \times S_m$ and $s_0 = (s_{01}, s_{02}, ..., s_{0m})$ is the initial state; 2. $\mathcal{AP} = \bigcup_{i \in \{1,2,\ldots,m\}} \mathcal{AP}_i$; 3. $\mathcal{L}(s = (s_1, s_2, s_3, ..., s_m)) = \bigcup_{i \in \{1,2,\ldots,m\}} \mathcal{L}_i(s_i)$.*

*The transition relation for $T_S, T_A$ and $T_{SA}$ is defined as follows:*

- Synchronous composition $T_S$: *$\mathcal{R} \subseteq S \times S$ such that given $s = (s_1, s_2, s_3, ...s_m)$ and $t = (t_1, t_2, t_3, ...., t_m)$, $\mathcal{R}(s, t)$ iff $\forall\ i \in \{1, 2, ..., m\}$ we have $\mathcal{R}_i(s_i, t_i)$, i.e., every component $T_i$ make a transition.*
- Asynchronous composition $T_A$: *$\mathcal{R} \subseteq S \times S$ such that given $s = (s_1, s_2, s_3, ...s_m)$ and $t = (t_1, t_2, t_3, ...., t_m)$, $\mathcal{R}(s, t)$ iff $\exists\ i \in \{1, 2, ..., m\}$ we have $\mathcal{R}_i(s_i, t_i)$, i.e., one or more component $T_i$ make a transition.*
- Strict asynchronous composition $T_{SA}$: *$\mathcal{R} \subseteq S \times S$ such that given $s = (s_1, s_2, s_3, ...s_m)$ and $t = (t_1, t_2, t_3, ...., t_m)$, $\mathcal{R}(s, t)$ iff for some $i$ we have $\mathcal{R}_i(s_i, t_i)$ and for all $j$ such that, $j \neq i$, we have $s_j = t_j$, i.e., exactly one of the components is allowed to make a transition.* ∎

We now present the syntax and semantics of CTL [2, 7].

**Syntax of CTL.** The syntax of CTL is as follows:

$S ::= \ p \mid \neg S \mid S \wedge S \mid AX(S) \mid EX(S) \mid A(S\ U\ S) \mid E(S\ U\ S)$ where $p \in \mathcal{AP}$.

In the syntax of ECTL the rules $AX(S)$ and $A(S\ U\ S)$ are not allowed. Similarly, in ACTL the rules $EX(S)$ and $E(S\ U\ S)$ are not allowed.

**Semantics of CTL.** The semantics of CTL is as follows:
• $s_0 \models p$ iff $p \in \mathcal{L}(s)$; • $s_0 \models \neg f$ iff $s_0 \not\models f$;
• $s_0 \models f_1 \wedge f_2$ iff $s_0 \models f_1$ and $s_0 \models f_2$;
• $s_0 \models AX(f)$ iff for all states $t$ such that $\mathcal{R}(s,t)$, $t \models f$;

The semantics for $EX(f)$ and $E(f_1\ U\ f_2)$ is similar to the semantics of $AX(f)$ and $A(f_1\ U\ f_2)$ with the for all states and for all paths quantifier changed to there exists a state and there exists a path, respectively.

## 3 Complexity of Compositional CTL Model Checking

In this section we study the complexity of CTL model checking on the composition of a set of tree-like kripke structures. We show that CTL model checking is PSPACE hard by reducing the QBF, that is, the truth of *Quantified Boolean Formulas* (QBF) [6], to the model checking problem. A QBF formula is of the following form: $\phi = \exists x_1 \forall x_2 \exists x_3 .... \forall x_n . C_1 \wedge C_2 ... \wedge C_m$. In the formula $\phi$ all $C_i$'s are clauses which are disjunction of literals (variables or negation of variables). We restrict each clause to have exactly three distinct literals. QBF is PSPACE complete with this restriction [6]. We reduce the QBF problem to the model checking of CTL formulas on synchronous/asynchronous composition of tree-like kripke structures. We present our reduction in steps as follows. We first present the idea of constructing a tree-like kripke structure for a given clause. Given a QBF formula $\phi$ with $m$ clauses $C_1, C_2, \ldots C_m$ we construct $m$ tree-like kripke structures $T_1, T_2, ..., T_m$, one for each clause. We define a CTL property $\psi$ on the composition of $T_1, T_2, ..., T_m$ (denoted as $T_S$) and show that $\psi$ is true in the start state of $T_S$ iff the QBF formula $\phi$ is true.

### 3.1 Construction of Tree-Like Kripke Structure for a Clause

Let $\phi$ be a QBF formula on a set of variables $X = \{x_1, x_2, ..., x_n\}$ and a clause $C_j$ with exactly three variables from $X$. We construct a tree-like kripke structure $T_j$, where $T_j$ is a tree of depth $n$, as follows:

1. The root of the tree $T_j$ is at depth 0. The root is the initial state of $T_j$.
2. If a node is at depth $i$ then the depth of its child (successor) is $i+1$.
3. A node $s$ at level $i$ has two children if variable $x_{i+1}$ appears in $C_j$, else $s$ has only one child.
4. The root is marked by the atomic proposition $r_j$.
5. If a variable $x_i$ appears in $C_j$ then for a node $s$ at depth $i$ then: if $s$ is a left child of its parent it is labeled with $p_{ji0}$, and if $s$ is a right child of its parent it is labeled with $p_{ji1}$.

6. If $x_i$ does not appear in $C_j$ then every node at depth $i$ is labeled with both $p_{ji0}$ and $p_{ji1}$.

The number of nodes at level $n$ is 8 for any tree as exactly three variables occur in any clause. Let the variables for the clause $C_j$ be $x_i, x_k, x_t$ where $i < k < t$. Let the nodes at the level $n$ be numbered from 0 to 7 in order as they will appear in an in-order traversal of $T_j$. The nodes at depth $n$ are labeled with the proposition $t_j$ as follows : Consider a node $s$ at depth $n$ such that it is numbered $i$. Let $B_1B_2B_3$ be the *binary representation* of $i$. If assigning $x_i = B_1, x_k = B_2, x_l = B_3$ makes $C_j$ false then $s$ is not labeled by $t_j$, otherwise it is labeled by $t_j$. ($B_1, B_2, B_3$ are $0, 1$ respectively and 0 represents false and 1 represents true). Intuitively the idea is as follows: the assignment of truth value 0 to variable $x_{i+1}$ in $C_j$ is represented by the choice of the left child at depth $i$ in $T_j$ and right child represents the assignment of truth value 1. We refer to the tree-like kripke structure for clause $C_j$, denoted by $T_j$, as the *clause tree kripke structure* for clause $C_j$. The tree structure corresponding to a clause is illustrated in the Figure 1.

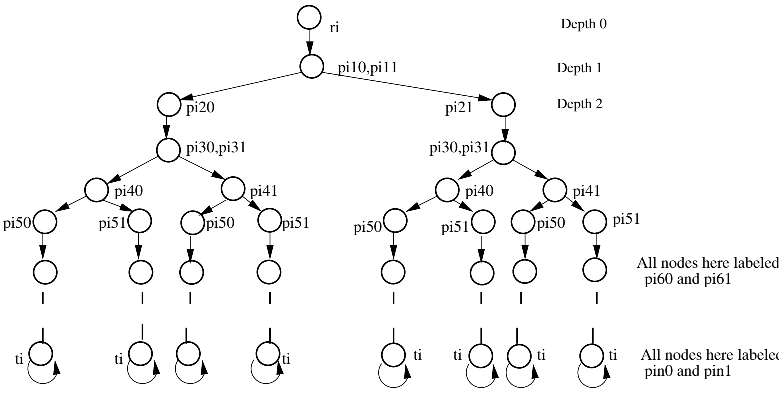

**Fig. 1.** Tree-like kripke structure for clause $C_i = (x_2 \vee \neg x_4 \vee x_5)$

The next Lemma follows from construction of tree-like kripke structures.

**Lemma 1.** *Let variables $x_i, x_k, x_t$ occur in $C_j$. Given a truth assignment to variables $x_i, x_k, x_t$ we can construct a path (state sequence) $(s_{j0}, s_{j1}, ..., s_{jn})$ in $T_j$ such that $s_{jn}$ is marked with $t_j$ iff the truth assignment makes $C_j$ true. The state sequence is constructed as follows:*

- *If $x_\ell$ is assigned false then $s_{j\ell}$ is the left child of $s_{j,\ell-1}$ and if $x_\ell$ is assigned true then $s_{j\ell}$ is the right child of $s_{j,\ell-1}$, where $\ell \in \{ i, k, t \}$.*

### 3.2 CTL Model Checking of Synchronous Composition

Given $m$ clauses $C_1, C_2, ..., C_m$ we construct the corresponding tree-like kripke structures $T_1, T_2, ..., T_m$ for the respective clauses. The synchronous composition of the tree-like structures is denoted as $T_S$. We define the properties

$p_i, 1 \leq i \leq n$ as follows : $p_1 = (\wedge_{j=1}^{m} p_{j10}) \vee (\wedge_{j=1}^{m} p_{j11})$, $p_2 = (\wedge_{j=1}^{m} p_{j20}) \vee (\wedge_{j=1}^{m} p_{j21})$ and in general, $p_i = (\wedge_{j=1}^{m} p_{ji0}) \vee (\wedge_{j=1}^{m} p_{ji1})$. Given a QBF formula with $m$ clauses an *inconsistent* assignment of truth value to a variable $x_i$ occurs if different truth values are assigned to variable $x_i$ in different clauses.

**Lemma 2.** *Consider a state sequence $\langle \nu_0, \nu_1, .....\nu_n \rangle$ in $T_S$ where $\nu_0 = s_0$ and each $\nu_i$ is the immediate successor of $\nu_{i-1}$. Let $\nu_i$ be represented as $(s_{i1}, s_{i2}, ..., s_{im})$. Let $O_i = \{k \mid x_i$ occurs in $C_k\}$. Then $\nu_i \models p_i$ iff one of the following conditions are satisfied:*

1. *for all $k \in O_i$ we have $s_{ik}$ is the left child of $s_{i-1,k}$.*
2. *for all $k \in O_i$ we have $s_{ik}$ is the right child of $s_{i-1,k}$.*

*Proof.* We prove the result considering the following cases:

1. If for all $k \in O_i$ we have $s_{ik}$ is the left child of $s_{i-1,k}$ in $T_k$ then $\nu_i \models \wedge_{j=1}^{m} p_{ji0}$. This is because for any clause $C_k$ in which $x_i$ occurs a node at depth $i$ in $T_k$ which is a left child satisfies $p_{ki0}$. In any clause $C_l$ such that $x_i$ does not occur a node depth $i$ is the only child of its parent in $T_l$ and satisfies $p_{li0}$. Similar argument can show that if for all $k \in O_i$ we have $s_{ik}$ is the right child of $s_{i-1,k}$ in $T_k$ then $\nu_i \models \wedge_{j=1}^{m} p_{ji1}$.
2. If $\exists\ t, l$ such that $t, l \in O_i$ and $s_{it}$ is the left child of $s_{i-1,t}$ in $T_t$ and $s_{il}$ is the right child of $s_{i-1,l}$ in $T_l$ then $\nu_i \not\models \wedge_{j=1}^{m} p_{ji0}$ as $\nu_i \not\models p_{li0}$ and $\nu_i \not\models \wedge_{j=1}^{m} p_{ji1}$ as $\nu_i \not\models p_{ti1}$. Intuitively, the state $\nu_i$ which does not satisfy $p_i$ actually shows that in one component, $T_t$, the left branch is followed at depth $i-1$ (representing the assignment of truth value 0 to $x_i$ in $C_t$) and in other component, $T_l$, the right branch is followed at depth $i-1$ (representing the assignment of truth value 1 to $x_i$ in $C_l$) which is represents a inconsistent truth value assignment to $x_i$. ∎

Given a QBF formula : $\phi = \exists x_1 \forall x_2 \exists x_3 .... \forall x_n . C_1 \wedge C_2 ... \wedge C_m$ where each $C_j$ is a clause with exactly three variables, for every clause we construct tree-like kripke structure as described in Section 3.1. Given the clauses $C_1, C_2, ..., C_m$ we have $T_1, T_2, ..., T_m$ as the respective tree-like kripke structures. Let $T_S$ denote the parallel synchronous composition of the $m$ kripke structures. We consider model checking the CTL property $\psi$ on $T_S$, where $\psi$ is defined as follows:

$$EX(p_1 \wedge AX(\neg p_2 \vee (p_2 \wedge EX(p_3 \wedge AX(\neg p_4 \vee (p_4 \wedge .... \\ EX(p_{n-1} \wedge AX(\neg p_n \vee (p_n \wedge (t_1 \wedge t_2 .... \wedge t_m)) \ldots)))))))).$$

We will prove that $\phi$ is true iff $\psi$ is true in the start state of $T_S$.

*Example 1.* We illustrate the whole construction through a small example. Given the following QBF formula $\phi_1$ with four variables and two clauses, the corresponding formula $\psi_1$ is as follows:

$$\phi_1 = \exists x_1 \forall x_2 \exists x_3 \forall x_4 . [(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)]$$

$$\psi_1 = EX(p_1 \wedge AX(\neg p_2 \vee (p_2 \wedge EX(p_3 \wedge AX(\neg p_4 \vee (p_4 \wedge (t_1 \wedge t_2)))))))$$

Given the clauses the corresponding tree-like kripke structures are shown in the figures, Figure. 2 and Figure. 3. ∎

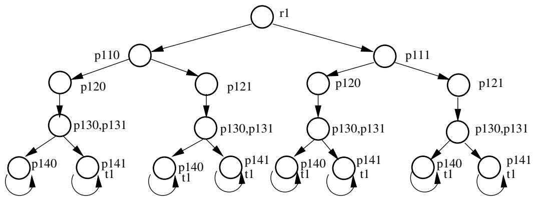

**Fig. 2.** The tree corresponding to the clause $(x_1 \vee x_2 \vee x_4)$

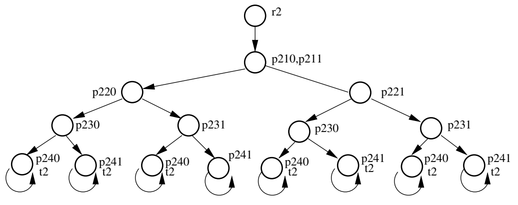

**Fig. 3.** The tree corresponding to the clause $(x_2 \vee \neg x_3 \vee \neg x_4)$

**Solution tree.** Given a QBF formula $\phi$ that is true there is a solution tree defined as follows to prove that $\phi$ is true. The solution tree can be described as:

- A node at depth $2*i$ has a one child that represents a truth value assignment of 0 or 1 assigned to $x_{2*i+1}$.
- A node at depth $2*i-1$ has two children: left child represents the truth value 0 and the right child represents the truth value 1 assigned to $x_{2*i}$.

A path from the root to a leaf represents an assignment of truth values to each variable $x_1, x_2, ..., x_n$. A solution tree proves the truth of $\phi$ iff for all paths from the root to a leaf the corresponding truth assignment satisfy each clause in $\phi$.

**Lemma 3.** *If $\phi$ is true then $\psi$ is true in the start state of $T_S$.*

*Proof.* Given a truth value assignment for variables $x_1, x_2, ..., x_n$ we follow the state sequence $(\nu_0, \nu_1, \nu_2, ...., \nu_n)$ with $\nu_0 = s_0$ as follows : ($\nu_i$ is represented as $(s_{i1}, s_{i2}, ..., s_{im})$ )

- If $x_i$ is assigned true then in all clauses $C_k$ where $x_i$ occurs $s_{ik}$ is the right child of $s_{i-1,k}$, else if $x_i$ is false then $s_{ik}$ is the left child of $s_{i-1,k}$. In all the other clauses $C_l$ in which $x_i$ does not occur $s_{il}$ is the only child of $s_{i-1,l}$.

It follows from Lemma 2 that in this state sequence $\nu_i$ satisfies $p_i$ . It also follows from Lemma 1 that $\nu_n$ satisfies $t_j$ iff the valuation of the variables makes $C_j$ true. Given a solution tree to prove $\phi$ we construct a proof tree $P_T$, that proves that $\psi$ holds in the start state of $T_S$, as follows:

1. For a node $\nu_{2*i}$ at depth $2*i$ in $P_T$ its immediate successor is defined as follows:
   - If $\nu_{2*i}$ satisfies $\neg p_{2*i}$ it has no successor.

  - Else $\nu_{2*i}$ satisfies $p_{2*i}$ and if $x_{2*i+1}$ is assigned true then in all $T_j$'s such that $x_{2*i+1}$ is in $C_j$ choose the right child in $T_j$ and if $x_{2*i+1}$ is assigned false then choose the left branch in $T_j$. In all $T_r$'s such that $x_{2*i+1}$ is not in $C_r$ choose the only immediate successor in $T_r$.

2. For a node at depth $2 * i + 1$ in $P_T$ its immediate successors are all its successors present in $T$.

Note that for any node at any depth $i$ only two of its successor can satisfy $p_{i+1}$, one representing the assignment of truth value 0 to $x_{i+1}$ where in all components left branches are taken and the other representing the assignment of truth value 1 to $x_{i+1}$ where in all components right branches are taken. A proof tree has been sketched in It follows that in $P_T$ a node at depth $2 * i + 1$ from the start state will satisfy $p_{2*i+1}$. Hence the node at depth $2 * i$ will satisfy $EX(p_{2*i+1})$. For a node at depth $(2 * i - 1)$ if in all the $T_j$'s for clause $C_j$ in which $x_{2*i}$ occurs left branches or right branches are followed (consistently in all $T_j$'s ) then the next state satisfy $p_{2*i}$. All the other successors satisfy $\neg p_{2*i}$. By the construction of $P_T$ it follows any leaf node which is at a depth $i$ where $i < n$ it satisfies $\neg p_i$. Since for the given solution tree $C_1 \wedge C_2 \wedge .... \wedge C_m$ is satisfied any leaf at depth $n$ will either satisfy $\neg p_n$ or will satisfy $t_1 \wedge t_2 \wedge ...t_m$. Hence $P_T$ proves that $\psi$ is satisfied in the start state of $T_S$. ∎

**Lemma 4.** *If $\psi$ is true in the start state of $T_S$ then $\phi$ is true.*

*Proof.* If the formula $\psi$ is true in the starting state of $T_S$ then there is a proof tree $P_T$ to prove $\psi$ to be true. We construct a solution tree to prove $\phi$. For a node $\nu_{2*i} = (s_{2*i,1}, s_{2*i,2}, ..., s_{2*i,m})$ at depth $2 * i$ in $P_T$ let $\nu_{2*i+1} = (s_{2*i+1,1}, s_{2*i+1,2}, ..., s_{2*i+1,m})$ be its successor such that $p_{2*i+1}$ is satisfied at $\nu_{2*i+1}$. From Lemma 2 we have that one of the following two conditions hold:

1. in every $T_j$ such that $x_{2*i+1}$ occurs in $C_j$, $s_{2*i+1,j}$ is the left child of $s_{2*i,j}$
2. in every $T_j$ such that $x_{2*i+1}$ occurs in $C_j$, $s_{2*i+1,j}$ is the right child of $s_{2*i,j}$.

If the former condition is satisfied then we construct the solution tree assigning truth value 0 (false) to $x_{2*i+1}$ and if the later is satisfied then we assign the value 1 (true) to $x_{2*i+1}$. As $\psi$ is true in the start state it follows that in $P_T$ any leaf at depth $n$ which satisfy $p_n$ must satisfy $t_1 \wedge t_2 ... \wedge t_m$. Hence the choice of the truth values for the odd variable as constructed above from the proof tree $P_T$ ensures that the solution tree thus constructed will prove $\phi$ (will satisfy all clauses). Hence if $\psi$ is true in the start state then $\phi = \exists x_1 \forall x_2 \exists x_3 .... \exists x_{n-1} \forall x_n .[C_1 \wedge C_2 ... \wedge C_m]$ is true. ∎

It follows from Lemma 3, 4 that the CTL model checking is PSPACE-hard. A DFS model checking algorithm that performs on-the-fly composition requires space polynomial in the size of the depth of the proof tree. This gives us the following result.

**Theorem 1.** *CTL model checking of synchronous composition of tree-like kripke structures is PSPACE complete.*

**Theorem 2.** *Model checking of formulas of the form $E(B\ U\ B)$ and $A(B\ U\ B)$, where $B$ is an Boolean formula, is NP complete and coNP complete respectively, for synchronous composition of tree-like kripke structures.*

*Proof.* Given a SAT formula in CNF (Conjunctive Normal Form) $\psi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ where each $C_i$ is a clause with exactly three variables from the set of variables of $\{x_1, x_2, \ldots, x_n\}$. For each clause $C_j$ we construct a clause tree-like kripke structure $T_j$ as described in Subsection 3.1. Let synchronous composition of the component kripke structures be $T_S$. We will prove that the SAT formula $\psi$ is satisfiable iff the following formula $\varphi$ is true in the start state of $T_S$, where $\varphi$ is defined as $\varphi = E(r \vee p_1 \vee p_2 \vee \ldots. \vee p_n U(t_1 \wedge t_2 \wedge \ldots \wedge t_m))$, where $r = r_1 \wedge r_2 \wedge \ldots \wedge r_n$. Note that for every $T_j$ the root of $T_j$ is marked with proposition $r_j$. Hence the staring state of $T$ will satisfy $r$.

Suppose $\psi$ is satisfiable, then there is a satisfying assignment $A$. Given $A$ we construct the following path (state sequence) $\nu_0, \nu_1, \nu_2, \ldots, \nu_n$ ,where $\nu_i = (s_{i1}, s_{i2}, \ldots, s_{im})$, to satisfy $\varphi$. We construct the immediate successor $\nu_i$ of $\nu_{i-1}$ as follows:

- if $x_i$ is assigned false by $A$ then in all $T_j$ such that $x_i$ occurs in $C_j$ the left branch is followed.
- if $x_i$ is assigned true by $A$ then in all $T_j$ such that $x_i$ occurs in $C_j$ the right branch is followed.

It is evident that $\nu_i$ satisfies $p_i$ (from Lemma 2) and $\nu_0$ satisfies $r$. Since $\psi$ is satisfiable we have $\nu_n$ satisfies $t_1 \wedge t_2 \wedge \ldots \wedge t_m$. So $\varphi$ is true in the start state.

If $\varphi$ is true at the start state then there is a path $P$ in $T$ to satisfy $(r \vee p_1 \vee p_2 \vee \ldots. \vee p_n U(t_1 \wedge t_2 \wedge \ldots \wedge t_m))$. Let the path be $\nu_0, \nu_1, \nu_2, \ldots, \nu_n$. Then in this path $\nu_i$ must satisfy $p_i$. For a node $\nu_i = (s_{i1}, s_{i2}, \ldots, s_{im})$ at depth $i$ in $P$ let $\nu_{i+1} = (s_{i+1,1}, s_{i+1,2}, \ldots, s_{i+1,m})$ be its successor such that $\nu_{i+1}$ satisfy $p_{i+1}$. It follows from Lemma 2 then one of the following two conditions must hold:

- in every $T_j$ such that $x_{2*i+1}$ occurs in $C_j$, $s_{2*i+1,j}$ is the left child of $s_{2*i,j}$.
- in every $T_j$ such that $x_{2*i+1}$ occurs in $C_j$, $s_{2*i+1,j}$ is the right child of $s_{2*i,j}$.

If the former condition is satisfied then assign $x_{i+1}$ to be 0 (false) and if the later is satisfied assign $x_{i+1}$ to be 1 (true). As $t_1 \wedge t_2 \wedge \ldots t_m$ is satisfied in the last state we have $\psi$ satisfied for the given assignment. This proves that the model checking of a simple formula of the form $E(B\ U\ B)$ is NP hard.

To prove the model checking is in NP we note that in $T$ any infinite path is path from the start state which is a state sequence of the form : $\nu_0, \nu_1, \nu_2, \ldots., \nu_i, \nu_i, \nu_i, \ldots..$ where $i$ is bounded by the maximum of the depth of the component tree-like kripke structure. Hence for any infinite path of the form: $\nu_0, \nu_1, \nu_2, \ldots., \nu_i, \nu_i, \nu_i, \ldots..$ which satisfies $E(B\ U\ B)$, $(\nu_0, \nu_1, \nu_2, \ldots., \nu_i)$ can be a proof. This proof is polynomial in size of the input. A NP algorithm guesses the state sequence and then verifies that the state sequence satisfies the formula $E(B\ U\ B)$, which can be achieved in $P$. The desired result follows.

To prove that the model checking problem is coNP hard for formulas of the form $A(B\ U\ B)$ we reduce the *validity* problem to it. Consider the problem of *validity* of a formula $\psi$ expressed in DNF (Disjunctive Normal Form) as follows: $\psi = F_1 \vee F_2 \vee ... \vee F_m$ where each $F_i$ is a *term* (conjunction of literals) with exactly three variables from the set of variables of $\{x_1, x_2, ..., x_n\}$. We construct the clause tree-like kripke structure $T_j$ for every term $F_j$ as mentioned in Section 3.1. The only difference is that every node in $T_j$ at depth $i$ is marked with a proposition $d_i$. Also the nodes at depth $n$ are marked with $t_i$ according to the following condition: Consider a node $s$ at depth $n$ such that it is numbered $i$. Let $B_1 B_2 B_3$ be the *binary representation* of $i$. If assigning $x_i = B_1, x_k = B_2, x_l = B_3$ makes $F_j$ true then $s$ is labeled by $t_j$, otherwise it is not labeled by $t_j$. ($B_1, B_2, B_3$ are $0, 1$ respectively and 0 represents false and 1 represents true). Let the synchronous composition of $T_1, T_2, \ldots, T_M$ be $T_S$. Consider the formula:

$$\varphi = A(r \vee p_1 \vee p_2 ... \vee p_n \; U \; (t_1 \vee t_2 \vee ... \; t_n) \vee (d_1 \wedge \neg p_1) \vee (d_2 \wedge \neg p_2) ... \vee (d_n \wedge \neg p_n))$$

where $r = r_1 \wedge r_2 \wedge ... \wedge r_n$. Similar argument as above with minor modifications for the *universal* nature of the $A$ operator and the validity problem we can show $\varphi$ is true in the start state of $T_S$ iff the formula $\psi$ is valid. The proof of the model checking problem of formulas of the form $A(B \; U \; B)$ is in coNP is similar. ∎

### 3.3 CTL Model Checking of Asynchronous Composition

Given $m$ clauses $C_1, C_2, ..., C_m$ we construct $T_1, T_2, ..., T_m$ as $m$ tree-like kripke structures for the respective clauses. Let $T_A$ denote the asynchronous composition of the tree-like structures. We prove that the CTL model checking of asynchronous composition is PSPACE complete. In this section we refer to $p_i$'s, $\psi, \phi$ as defined in the Subsection 3.2. The construction of the tree-like kripke structure in Subsection 3.1 gives us the following result.

**Lemma 5.** *Given a state $\nu_i = (s_{i1}, s_{i2}, ....s_{in})$ in $T_A$ such that $\nu_i$ satisfies $p_k$ then for all $j$, depth of $s_{ij}$ in $T_j$ is $k$.*

**Theorem 3.** *CTL model checking of asynchronous composition of tree-like kripke structures is PSPACE complete.*

*Proof.* Consider the formula $\phi$ and $\psi$ as described in the Subsection 3.2. Consider the start state $s$ in $T_A$. It follows from Lemma 5 that any successor $s_1$ of $s$ in $T_A$ which satisfies $p_1$ follows from a transition in which all the components make a transition (which corresponds to a transition of the synchronous composition). Similarly consider any successor $s_2$ of $s_1$, a transition in which all the components does not make a transition will cause $s_2$ to satisfy $\neg p_2$. For a transition which satisfies $p_2$ it will have to be a transition in which all the component $T_j$'s make a transition (which again corresponds to a transition of the synchronous composition). This argument can be extended for any depth $2 * i$ and $2 * i + 1$. Hence the construction of the proof tree $P_T$ from a given solution tree of truth values to variables to prove $\psi$ and the construction of a solution tree from the proof tree $P_T$ is similar as in the Lemmas and Theorems in the Subsection 3.2. This proves that CTL model checking of $T_A$ is PSPACE hard. The PSPACE upper bound argument is similar to Theorem 1. ∎

Lemma 5, Theorem 3 and arguments similar to to Theorem 2 gives us the following Theorem.

**Theorem 4.** *Model checking of formulas of the from $E(B\ U\ B)$ is NP complete and model checking of formulas of the form $A(B\ U\ B)$ is coNP complete, where $B$ is a Boolean formula, for asynchronous composition of tree-like kripke structures.*

### 3.4 CTL Model Checking of Strict Asynchronous Composition

Given $m$ clauses $C_1, C_2, ..., C_m$ we construct $T_1, T_2, ..., T_m$ as $m$ tree-like kripke structures for the respective clauses. We denote by $T_{AS}$ the strict asynchronous composition of the tree-like structures. For every node $s$ in $T_j$ such that the depth of $s$ is $d$ it is marked with an atomic proposition $l_{jd}$. In this section we refer to $p_i$'s , $\phi$ as defined in the Subsection 3.2. We define properties $l_i, l'_i$ at a node in $T$, for $1 \leq i \leq n$ as follows: $l_i = \wedge_{j=1}^{m}(l_{ji} \vee l_{j,i-1})$, $l'_i = \wedge_{j=1}^{m}(l_{ji})$. We define $\psi$ as follows:

$$\psi = E(l_1 U(p_1 \wedge A(l_2 U(\neg l_2 \vee (\neg p_2 \wedge l'_2) \vee (p_2 \wedge E(l_3 U(p_3 \wedge \\ ...A(l_n U(\neg l_n \vee (\neg p_n \wedge l'_n) \vee (p_n \wedge (t_1 \wedge t_2 ... \wedge t_m))))...)))))))$$

We briefly sketch the idea of the proof of the reduction of QBF to CTL model checking of strict asynchronous composition. The property $l_i$ is true at a state if the depth of every component node is either $i$ or $i-1$. Consider a path $\pi = (s_0, s_1, ...)$ which satisfy $l_1 U p_1$, where $s_0$ is the start state of $T$. In the path $\pi$ in no component more than one transition is taken. When $p_1$ is reached all components must have taken one transition each. Hence it corresponds to a single transition of a synchronous composition. Consider a state which satisfy $s' \in T$ such that $s'$ satisfies $p_1$. The state $s'$ is a state in $T_{AS}$ such that depth of all the component nodes is 1. We consider the truth of the formula $A(l_2 U(\neg l_2 \vee (l'_2 \wedge \neg p_2) \vee p_2)$ in $s'$. The part $(\neg l_2 \vee (l'_2 \wedge \neg p_2))$ ensures the following :

- If there is more than one transition in a component then $\neg l_2$ is satisfied.
- If in all components one transition is made and there are components such that in one component the left branch transition is followed whereas in the other component the right branch transition is followed then we have $l'_2 \vee \neg p_2$ satisfied.

So in the above cases $A(l_2 U(\neg l_2 \vee (l'_2 \wedge \neg p_2) \vee p_2)$ cannot be false. So any $l_2$ path to a state with more than one transition for any component or to a state which is a representative of inconsistent truth values to variable $x_2$ (a state which satisfy $\neg p_2$) in different clauses will not cause $A(l_2 U(\neg l_2 \vee (l'_2 \wedge \neg p_2) \vee p_2)$ to be falsified. A $l_2$ path to a state satisfying $p_2$ again corresponds to a single synchronous transition. Similar arguments can be extended to depth $2 * i$ and $2 * i + 1$ respectively. The rest follows arguments similar to those in Lemmas and Theorems in the Subsection 3.2 and 3.3 to prove that the model checking of strict asynchronous composition of tree-like kripke structure is PSPACE complete.

**Theorem 5.** *CTL model checking of strict asynchronous composition of tree-like kripke structures is PSPACE complete.*

*Remark 1.* The PSPACE-upper bound for CTL model checking holds for synchronous, asynchronous and strict asynchronous composition even if the component structures are arbitrary kripke structure. (i.e., underlying transition relation is a graph rather than a tree).

## 4 Reachability Analysis

The reachability problem asks given two states $s$ and $t$ in the composition of $m$ tree-like kripke structure whether there is a path from $s$ and to $t$.

**Synchronous Composition.** Let $s = (s_1, s_2, ..., s_m)$ and $t = (t_1, t_2, ..., t_m)$ be two states. It can be shown that $t$ is reachable from $s$ if and only if the following two conditions hold: (a) for all non-leaf nodes $t_i$ and $t_j$ we have $depth(t_j) - depth(s_j) = depth(t_i) - depth(s_i) = d$, and (b) for all leaf nodes $t_k$ we have $depth(t_k) - depth(s_k) \leq d$. The values for $depth(t_i) - depth(s_i)$ can be computed by a simple BFS algorithm linear in the size of the input.

**Theorem 6.** *Given two states $s = (s_1, s_2, ..., s_m)$ and $t = (t_1, t_2, ..., t_m)$ whether $t$ is reachable from $s$ can be determined in time linear in the input size for synchronous composition of tree-like kripke structures.*

**Asynchronous Composition.** For asynchronous and strict asynchronous composition reachability analysis is linear even if the individual components are arbitrary kripke structures. Given $m$ kripke structures $G_1, G_2, ..., G_m$ let $G$ be their asynchronous composition (or strict asynchronous composition).

**Theorem 7.** *Given two states $s = (s_1, s_2, ..., s_m)$ and $t = (t_1, t_2, ..., t_m)$ whether $t$ is reachable from $s$ can be determined in time linear in the input size for asynchronous and strict asynchronous composition of arbitrary kripke structures.*

## References

1. Burch, J.R., Clarke, E.M., Long, D.E., McMillan, K.L., and Dill, D.L., Symbolic model checking for sequential circuit verification. *IEEE Trans. on Computer Aided Design*, **13**, 4, 401-424, 1994.
2. Clarke, E.M., Emerson, E.A., and Sistla, A.P., Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Program. Lang. & Systems*, **8**, 2, 244-263, 1986.
3. Clarke, E.M., and Kurshan, R.P., Computer aided verification. *IEEE Spectrum*, **33**, 6, 61-67, 1996.
4. Cormen, T.H., Leiserson, C.E., and Rivest, R.L., *Introduction to Algorithms.* The MIT Press, Cambridge and McGraw-Hill, 1990.
5. Emerson, E.A., Mok, A.K., Sistla, A.P. and Srinivasan, J., Quantitative temporal reasoning, In *First Annual Workshop on Computer-Aided Verification*, France, 1989.
6. Papadimitriou, C.H., *Computational Complexity*, Addison-Wesley, 1994.
7. Clarke, E.M., Grumberg, O., and Peled, D.A., *Model Checking*, MIT Press, 2000.