Treewidth in Verification: Local vs. Global*

Andrea Ferrara[1], Guoqiang Pan[2], and Moshe Y. Vardi[2]

[1] DIS - Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
Email: ferrara@dis.uniroma1.it
[2] Dept. of CS, MS 132, Rice University,
6100 Main St.,
Houston TX 77005, USA
Email: {gqpan,vardi}@cs.rice.edu

The *treewidth* of a graph measures how close the graph is to a tree. Many problems that are intractable for general graphs, are tractable when the graph has bounded treewidth. Recent works study the complexity of model checking for state transition systems of bounded treewidth. There is little reason to believe, however, that the treewidth of the state transition graphs of real systems, which we refer to as *global* treewidth, is bounded. In contrast, we consider in this paper *concurrent* transition systems, where communication between concurrent components is modeled explicitly. Assuming boundedness of the treewidth of the communication graph, which we refer to as *local* treewidth, is reasonable, since the topology of communication in concurrent systems is often constrained physically.

In this work we study the impact of local treewidth boundedness on the complexity of verification problems. We first present a positive result, proving that a CNF formula of bounded treewidth can be represented by an OBDD of polynomial size. We show, however, that the nice properties of treewidth-bounded CNF formulas are not preserved under existential quantification or unrolling. Finally, we show that the complexity of various verification problems is high even under the assumption of local treewidth boundedness. In summary, while global treewidth boundedness does have computational advantages, it is not a realistic assumption; in contrast, local treewidth boundedness is a realistic assumption, but its computational advantages are rather meager.

1 Introduction

The *treewidth* of a graph measures how close the graph is to a tree (trees have treewidth 1). Many problems that are intractable (e.g. NP-hard, PSPACE-hard) for general graphs, are polynomial or linear-time solvable when the graph has bounded treewidth (see [5–7] for an overview). For example, constraint-satisfaction problems, which are NP-complete, are PTIME-solvable when the variable-relatedness graph has bounded treewidth [11, 14].

In [15, 22] the complexity of the model-checking problem is studied under the hypothesis of bounded treewidth; that is, it is assumed that the model is a state transition

* Work supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, IIS-9908435, IIS-9978135, EIA-0086264, and ANI-0216467, by BSF grant 9800096, by Texas ATP grant 003604-0058-2003, and by a grant from the Intel Corporation.

system, whose underlying graph has bounded treewidth. Bounding treewidth yields a large class of tractable model-checking problems. For example, while it is not known whether model checking μ-calculus formulas is in PTIME [18], it is in PTIME under the bounded treewidth assumption [22].

We refer to the treewidth of the state transition graphs of transition systems as the *global treewidth*. The global treewidth-boundedness assumption used in [15, 22] is not, in our opinion, useful to describe real-world verification problems. There is little reason to believe that the global treewidth of real-world systems is bounded. For example, it is easy to see that the graphs underlying systems with two counters are essentially grids, which are known to have high treewidth [26]. In verification practice, real-world systems are often modeled as *concurrent* transition systems, where communication between concurrent components is modeled explicitly. When we consider the communication graph between the concurrent components (the component are the nodes, and an edge exists between each pair of communicating nodes), assuming treewidth boundedness is not unreasonable. Indeed, the topology of communication in concurrent systems is often constrained physically; for example, by the need to layout a circuit in silicon. Such topological constraints are studied, for example, in [20, 23]. In [20] the width of a Boolean circuit is related to the size of its corresponding OBDD, while in [23] bounded cutwidth is used to explain why ATPG, an NP-complete verification problem, is so easy in practice. Cutwidth boundedness is used also to improve symbolic simulation and Boolean satisfiability in [4, 29]. These various notions of bounded width are assumed because of the constrained topology of communication in concurrent systems.

In this paper, we refer to treewidth of the component communication graph as *local* treewidth and study the impact of local-treewidth boundedness on the complexity of verification problems. We believe that because the component communication graph is often constrained physically, as noted above, assuming local treewidth boundedness is natural and realistic. (In fact, the assumption of treewidth boundedness is less severe than related assumption that are often made, such as *pathwidth* boundedness or *cutwidth* boundedness [5–7].)

We first present a positive result. We prove that a CNF formula of bounded treewidth can be represented by an OBDD of polynomial size (treewidth here is defined on the primal graph of the formula, where vertices represent variables and edges represent the co-occurance of the variables in the same clause). Thus, if a transition relation of a concurrent transition system is specified by a CNF formula with bounded treewidth, then there is an OBDD of polynomial size representing it. In contrast, the OBDD of transition relations often blow up, requiring symbolic model-checking techniques that avoid building these OBDDs [2].

We then show that bounded local treewidth offers little computational advantage for verification in general. First, we show that the small-OBDD property of bounded treewidth CNF formulas is destroyed as soon as we apply existential quantification, which is a basic operation in symbolic model checking, since the image operations involves existential quantification [20]. We then show that treewidth boundedness of a transition relation is not preserved under unrolling, which is a basic operation in SAT-based bounded model checking (BMC) [3]. (Note that while satisfiability of CNF formulas is NP-complete, satisfiability of bounded-treewidth CNF formulas can be solved in polynomial time, cf. [1]).

Finally, we show that the complexity of various verification problems are high even under the assumption of local treewidth boundedness. We review several verification problem for concurrent systems, including model checking, simulation, and containment, and show that the known lower bounds (PSPACE-complete, EXPTIME-complete, and EXPSPACE-complete, respectively [16, 19]) hold also under the assumption of local treewidth boundedness. (Our results are robust: the lower bound apply even under pathwidth boundedness or cutwidth boundedness.)

In summary, while global treewidth boundedness does have computational advantages, it is not a realistic assumption. In contrast, local treewidth boundedness is a realistic assumption, but its computational advantages are rather meager.

The paper is organized as follows: In Section 2 we prove the small-OBDD property for transition relations of bounded treewidth, but then show that this property does not help in symbolic model checking and in bounded model checking. Finally, in Section 3 we show that lower bound for model checking, simulation, and containment hold also under the assumption of local treewidth boundedness.

2 Transition Relation: OBDDs size and BMC

The notions of treewidth and pathwidth were introduced in [25, 26].

Definition 2.1. *A tree decomposition of a graph $G = (V, E)$ is a pair (T, X), where $T = (I, F)$ is a tree whose node set is I and edge set is F, and $X = \{X_i | i \in I\}$ is a family of subsets of V, one for each node of T, such that:*

- $\bigcup_{i \in I} X_i = V$.
- *for every edges $(v, w) \in E$, there exists an $i \in I$ with $\{v, w\} \subseteq X_i$.*
- *for all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

The *width* of a tree decomposition (T, X) is $max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G is the minimum width over all possible tree decompositions of G. The notions of path decomposition and pathwidth are defined analogously, with the tree T in the tree decomposition restricted to be a path. By Corollary 24 in [7], we know that for a graph G with n vertices we have that $pathwidth(G) = O(treewidth(G) \cdot \log n)$. Clearly, $treewidth(G) \leq pathwidth(G)$.

Definition 2.2. *The Gaifman graph of a CNF formula is a graph having one vertex for each variable and an edge (v_1, v_2) if the variables v_1 and v_2 occur in the same clause of the formula. By treewidth (pathwidth) of a CNF formula we refer to the treewidth (pathwidth) of its Gaifman graph.*

Ordered Boolean decision diagrams (OBDDs) [8] are a canonical form representation for Boolean formulas. An OBDD is a rooted, directed acyclic graph with one or two terminal nodes labeled **0** or **1**, and a set of variable nodes of out-degree two. The variables respect a given linear order on all paths from the root to a leaf. Each path represents an assignment to each of the variables on the path. Since there can be exponentially more paths than vertices and edges, OBDDs can be substantially more compact than traditional representations like CNF. In many case, however, going from CNF representation to OBDD representation may cause an exponential blow-up [2]. We now show that this is not the case when the CNF formula has bounded treewidth.

Theorem 2.1. *A CNF formula C with n variables and pathwidth q has an OBDD of size $O(n2^q)$.*

Proof. Let the path decomposition of C be (P, L). Assume without loss of generality that $P = \{1, \ldots, k\}$. We construct a variable order from the path decomposition as follows: Define $First(x) = \min(\{p \in P \mid v \in L(p)\})$ and $Last(x) = \max(\{p \in P \mid v \in L(p)\})$. Now sort the variables in increasing lexicographic order according to $(First(x), Last(x))$; that is, define the variable order so that if $x < y$, then either $First(x) < First(y)$ or $First(x) = First(y)$ and $Last(x) < Last(y)$. We show that, using this variable order, there are at most 2^q nodes per level. The claim then follows.

For each clause c, we define $\min(c)$ as the index of the lowest ordered variable in c and correspondingly for $\max(c)$. Consider level i of the OBDD, corresponding to the variable x_i. The clause set C can be partitioned into three classes with respect to level i, $C_{ended} = \{c \mid \max(c) < i\}$, $C_{cur} = \{c \mid \min(c) \leq i < \max(c)\}$, and $C_{untouched} = \{c \mid i < \min(c)\}$.

A node u at level i corresponds to a set A_u of partial assignments to variables, where each partial assignment $a \in A_u$ is an element in $2^{\{x_1 \cdots x_{i-1}\}}$. For a partial assignment a and a clause set D, we write $a \models D$ if a is a model of D, i.e, for each clause $c \in D$, a satisfies some literal in c. From the semantics of OBDDs, we know that all partial assignments a in A_u are equivalent with respect to extensions, i.e., given $a' \in 2^{\{x_i,\ldots,x_n\}}$ and $a \in A_u$, we have that $a \cup a' \models C$ iff for every $a'' \in A_u$, $a'' \cup a' \models C$. If for $a \in A_u$, $a \not\models C_{ended}$, then we know that for every extension $a \cup a'$ of a we have that $a \cup a' \not\models C_{ended}$, so $a \cup a' \not\models C$. Thus, the node u is identical to Boolean 0 and should not exist at level i. It follows that for every $a \in A_u$, $a \models C_{ended}$. We also know that all clauses in $C_{untouched}$ have none of their variables assigned by $a \in A_u$.

Each partial assignment a at level i can be associated with a subset $M_a \subseteq C_{cur}$ where $M_a = \{c \mid c \in C_{cur}, a \models c\}$, i.e., the clauses in C_{cur} that are already satisfied by a before reading the variable x_i. We know that none of the clauses in C_{cur} have failed (all literals assigned to false) so far, since by definition of C_{cur} all such clauses have literals with variables beyond x_{i-1}. Suppose that for two distinct nodes u and v at level i there exists $a_u \in A_u$ and $a_v \in A_v$ such that $M_{a_u} = M_{a_v}$. Since u and v are distinct, there is a partial assignment $a \in 2^{\{x_i,\ldots,x_n\}}$ that distinguishes between u and v; say, $a_u \cup a \models C$ and $a_v \cup a \not\models C$. Since a_u and a_v, however, both satisfy C_{ended}, both are undefined on the variables of $C_{untouched}$, and we also have, by assumption, that $M_{a_u} = M_{a_v}$, we must have that $a_u \cup a \models C$ iff $a_v \cup a \models C$ – a contradiction. It follows that $M_{a_u} \neq M_{a_v}$.

Let $j = First(x_i)$. We know that $L(j)$ contains at most $q + 1$ variables, including x_i. Let $Var_i = L(j) \cap \{x_1, \ldots, x_{i-1}\}$, then Var_i has at most q variables. Suppose that u and v are two nodes at level i such that there exists $a_u \in A_u$ and $a_v \in A_v$ where

a_u and a_v agree on Var_i. We show then $M_{a_u} = M_{a_v}$. Consider a clause $c \in C_{cur}$. We know that all the variables of c occur in $L(k)$ for some k. We cannot have $k < j$, since then we'd have $c \in C_{ended}$, so $k \geq j$. If x_h occurs in c for some $h < i$, then by construction $x_h \in L(j')$ for some $j' \leq j$. By the property of path decompositions it follows that $x_h \in L(j)$. Since a_u and a_v agree on Var_i, it follows that they agree on c. We showed that if u and v are distinct, then for every $a_u \in A_u$ and $a_v \in A_v$, $M_{a_u} \neq M_{a_v}$. It follows that a_u and a_v cannot agree on Var_i. Since Var_i has at most q variables, there can be at most 2^q nodes at level i. The claim follows since the OBDD as n levels.

□

The relationship described in Theorem 2.1 between pathwidth and OBDD size was first shown in [17]. The proof there goes via a variant of a DPLL-based satisfiability algorithm. Our argument here is direct and show how to obtain an OBDD variable order from a path decomposition.

Recall that we know that for a graph G with n vertices we have that $pathwidth(G) = O(treewidth(G) \cdot \log n)$.

Corollary 2.1. *A CNF formula C with n variables and treewidth width q has an OBDD of size polynomial in n and exponential in q.*

While Theorem 2.1 suggests that OBDD-based algorithms are tractable on bounded width problems, typical model-checking algorithms do more than just build OBDDs that correspond to CNF formulas. OBDDs are often used to perform symbolic image operations, which requires applying existential quantification to OBDDs [20]. While it is often claimed that fixed-parameter tractability implies tractability for the bounded-parameter case, the constant factor resulting from the blowup of the parameter needs to be considered on a case-by-case basis. Often, super-exponential blowups in the parameter indicates that the problem is not practically tractable. The following theorem shows that Theorem 2.1 is not likely to be useful in model checking, since using quantification on bounded-width formulas leads to such a super-exponential blowup on the constant factor that is based on the parameter.

Theorem 2.2. *There exists a formula C in CNF with n variables and pathwidth q, and a subset of variables X such that $(\exists X)C$ under every variable order does not have a OBDD of size $n2^{f(q)}$, for a sub-exponential function f.*

Proof. We consider the hidden-weighted bit (HWB) function, which is shown in [9] to have a OBDD size of $\Omega(1.14^m)$ under arbitrary variable order, where m is the number of input bits. The HWB function is a Boolean function $2^m \rightarrow \{0, 1\}$, where for an m-bit input vector A, the output is the wth bit of A, w being the number of 1s in A (the *bit count* of A). The OBDD is defined on the set of variables $A[0]$ to $A[m-1]$.

We consider the case where $m = 2^k$, $k > 3$, and use a CNF formula to represent the HWB function. Clearly, from the upper bounds shown in Corollary 2.1, a direct translation can not result in bounded pathwidth; we use $(m+1)k+1$ additional existentially quantified variables to facilitate the CNF encoding. In the additional variables, there are $m+1$ counters (at k bits each), which we call $X_0, \ldots X_m$, and a single bit witness

w. Each X_i is used to guess the number of 1s occurring after $A[i]$. The bit witness w guesses the value of $A[X_0]$. We use CNF constraints to check the correctness of our guesses. The CNF formula C is the conjunction of all the following constraints. ($=$ and $+$ are short hand defined on bit vectors of size k):

- For each $0 \leq i < m$, we define $C_i^1 := (A[i] \rightarrow X_i = X_{i+1} + 1) \wedge (\neg A[i] \rightarrow X_i = X_{i+1})$. This asserts that if X_i is a correct guess iff X_{i+1} is a correct guess.
- For each $0 \leq i < m$, we define $C_i^2 := (X_0 = i) \rightarrow (A[i] \leftrightarrow w)$. This asserts that w is a correct guess if X_0 is a correct guess.
- $C^g := w$. Since we are building the OBDD representing inputs where the HWB function returns 1, w is asserted to true.
- The well-formedness constraint is $C^{wf} := X_m = 0$. This asserts that X_m is a correct guess. Combined with the C_i^1s, they assert that all X_is are correct guesses.

The only shorthand we used above is $=$ and $+$ on bit vectors of length k, both of which can be written out in CNF with no additional variables and $O(k^2)$ clauses. Now, $(\exists X_0) \ldots (\exists X_m)(\exists w)C$ characterizes the HWB function.

Next we show there is a path decomposition of C of width $3k+1$. There is one node per bit in A, ordered from 0 to $m-1$. Each node contains the support for the constraints C_i^1 and C_i^2 (the last node also contains C^g and C^{wf} with no additional variables). In turn, each node i contains the variables $A[i]$, w, X_0, X_i, and X_{i+1}, giving a pathwidth of $3k + 1$.

Consider the relationship between the size of the OBDD and the pathwidth. Assume we have a BDD of size $n2^{f(q)}$, where the pathwidth is q and the number of variables is n, and f is a sub-exponential function. Here, $q = 3k + 1$ and $n = (m + 1)k + 1 + m = (2^k + 1)(k + 1)$. The size of the OBDD S is then $((2^k + 1)(k + 1))2^{f(3k+1)} < 2^{(k+3)}2^{f(3k+1)} = 2^{f(3k+1)+k+3} = 2^{g(k)}$. Since f is sub-exponential, g is sub-exponential as well. But from [9], the lower bound for the size of such OBDDs is $\Omega(1.14^m) = \Omega(2^{\log\ 1.14 \times 2^k})$, which contradicts with g being sub-exponential. So such small OBDDs cannot exist.

□

Next we show that our construction is almost worst case, i.e., there is a closely related upper-bound.

Theorem 2.3. *For a CNF formula $C = \bigwedge c$ on n variables with pathwidth q and a subset of variables X, the formula $(\exists X)C$ has an OBDD of size $O((n - |X|)2^{2^q})$.*

Proof. To get the upper bound, we use the same approach as the Theorem 2.1, i.e., we show an upper bound of 2^{2^q} nodes for nodes at each level i by counting the number of equivalence classes.

We use $supp(C)$ to denote the set of variables that occur in C, and define $Y = supp(C) - X$ as the set of *free* variables in $(\exists X)C$. We use the same variable order as Theorem 2.1, and name the variables in Y as $y_1, y_2 \ldots y_m$ according to the variable order. For a set $Z \subseteq supp(C)$, we use $Z_{<i}$ to denote the subset that appears before y_i in the variable order. Also, Z_j is used to denote the subset of Z that occurs in path-decomposition node j. Each node u corresponding to a variable y_i represents a set of assignments A_u to $Y_{<i}$, encoded by the paths to the node from the root of the OBDD.

Consider an assignment $a \in A_u$. For each assignment $b \in 2^{X_{<i}}$ to the quantified variables occurring before y_i, we have a corresponding set of clauses in C that are satisfied by $a \cup b$. Assume that y_i occurs in node k of the path decomposition of C. Recall that C can be partitioned into C_{ended}, C_{cur}, and $C_{untouched}$ based on the variable y_i. Define the function $F_a : 2^{X_{k,<i}} \rightarrow \{\bot\} \cup 2^{C_{cur}}$ such that for each assignment b to $X_{k,<i}$, $F_a(b) = \bot$ if there is no extension b' (on $X_{<i}$) of b such that $a \cup b' \models C_{ended}$; otherwise, $F_a(b) = S$ where $S \subseteq C_{cur}$ is the clauses in C_{cur} satisfied by $a \cup b$. Now, we show that two distinct nodes u and v corresponding to y_i do not contain assignments a_u in A_u and a_v in A_v such that $F_{a_u} = F_{a_v}$. Assume the contrary. Since u and v are distinct, w.l.o.g., there is an assignment a to $Y_{\geq i}$ such that $a_u \cup a \models (\exists X)C$ and $a_v \cup a \not\models (\exists X)C$. Take an assignment b on X where $a_u \cup a \cup b \models C$. Let b' be a restriction of b to the variables in $X_k \cup X_{\geq i}$, and let b'' be a restriction of b to the variables in $X_{k,<i}$. It is clear that $a \cup b' \models C_{untouched}$. We know that $F_{a_u}(b'') \neq \bot$, since b restricted to $X_{<i}$, which we call b_{a_u}, satisfies $a_u \cup b_{a_u} \models C_{ended}$. Since $F_{a_u} = F_{a_v}$, $F_{a_v}(b'') = F_{a_u}(b'') \neq \bot$. Again, we have an extension b_{a_v} (from the definition of F_{a_v}) of b'' to $X_{<i}$ where $a_v \cup b_{a_v} \models C_{ended}$. For a clause $c \in C_{cur}$, if $c \in F_{a_u}(b'')$, then $c \in F_{a_v}(b'')$, so $a_v \cup b_{a_v} \models c$. Otherwise, $a \cup b' \models c$, since $a_u \cup a \cup b \models c$ and $a_u \cup b'' \not\models c$. So, $a_v \cup b_{a_v} \cup a \cup b' \models C_{cur}$. In summary, $a_v \cup a \cup b_{a_v} \cup b' \models C$, which contradicts with $a_v \cup a \not\models (\exists X)C$.

Now we count the number of possible functions for F_a. For each $b \in 2^{X_{k,<i}}$, the number of possible choices of $F_a(b)$ is $1 + 2^{|Y_{k,<i}|}$ since the satisfaction of clauses in C_{cur} depends only on b and assignments to $Y_{k,<i}$. Thus, the number of possible such F_as is $(1 + 2^{|Y_{k,<i}|})^{2^{|X_{k,<i}|}} \leq (2^{|Y_{k,<i}|+1})^{2^{|X_{k,<i}|}} = 2^{(|Y_{k,<i}|+1)2^{|X_{k,<i}|}} \leq 2^{2^q}$ since $q \geq |X_{k,<i}| + |Y_{k,<i}|$.

The combination of the possible count of F_as and the fact that distinct nodes induce distinct F_as gives us a bound of 2^{2^q} nodes at each level, i.e., a size bound of $(n-|X|)2^{2^q}$ for the whole OBDD.

□

The double exponential blowup for the OBDD size of quantified bounded pathwidth formulas on the pathwidth prevents us from using $pathwidth(G) = O(treewidth(G) \log n)$ to achieve a polynomial size OBDD for quantified bounded treewidth formulas. Whether a non-polynomial lower bound exists for the OBDD size of quantified bounded treewidth formulas is left for future research.

Let us now consider the effect of the local bounded treewidth on the complexity of Bounded Model Checking (BMC). In bounded model checking, variable substitutions are used to create distinct copies of the system. Given a formula f with support set $V = \{v_1, v_2, \ldots v_n\}$, and a substitution variable set $V' = \{v'_1, v'_2, \ldots v'_n\}$, we write $f[V/V']$ to represent a copy of f where each v_i in f is replaced with v'_i. To unroll a system to k iterations, we create $k+1$ copies of the state variable set V, which we call $V^0, V^1, \ldots V^k$. The transition relation is a formula over $V \cup V'$, where V is the current state variables and V' is the next state variables. The BMC *unrolling* would contain $\bigwedge_{0 \leq i \leq k-1} TR[V/V^i, V'/V^{i+1}]$, in addition to initial and property constraints. In the following theorem, we show that BMC unrolling does not preserve the bounded treewidth.

Theorem 2.4. *Even though the transition relation of a concurrent transition system, represented by a CNF $TR(V, V')$, has bounded treewidth, its unrolling can have unbounded treewidth.*

Proof. As an example, we take the case where the state variable set V is $\{x_1, x_2, \ldots x_w\}$ and the transition function is defined by $x'_i := (x_{i-1} \leftrightarrow x_i) \leftrightarrow x_{i+1}$. The CNF for transition relation $TR(V, V')$ clearly has bounded pathwidth (where each path decomposition node consists of the variables $x_i, x_{i+1}, x'_i, x'_{i+1}$), and, in turn, bounded treewidth.

Now we considering Gaifman graph of the unrolling. An example where two copies are unrolled is shown in Figure 1. The state variable for x_i at iteration j is denoted as x_i^j. We can see clearly that if we unroll, say, $w + 2$ copies, the Gaifman graph will have a $w \times w$ grid as a minor, which implies unbounded pathwidth (and treewidth) [12]. □

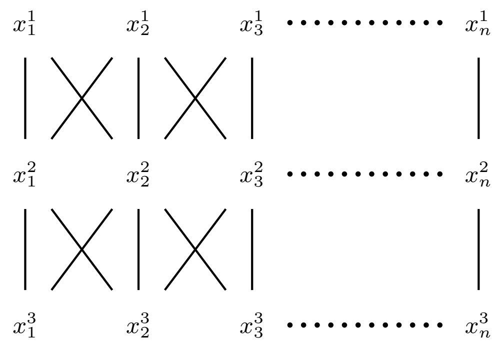

Fig. 1. The $TR(k)$ in Theorem 2.4, for $k = 3$

3 Model Checking, Containment, Simulation

We now introduce definitions of non-deterministic transition systems with bounded concurrency [16]. A non-deterministic transition system with bounded concurrency (*concurrent transition system* for short) is a tuple $P = \langle O, P_1, \ldots, P_n \rangle$ consisting of a finite set O of *observable events* and *n components* $P_1, \ldots, P_n$ for some $n \geq 1$. Each component P_i is a tuple $\langle O_i, W_i, W_i^0, \delta_i, L_i \rangle$, where:

- $O_i \subseteq O$ is a set of local observable events. The O_j are not necessarily pairwise disjoint; hence, observable events may be shared by several components. We require that $\bigcup_{j \in I}^n O_j = O$.
- W_i is a finite set of states, and we require that the W_j be pairwise disjoint. Also we let $W = \bigcup_{j \in I}^n W_j$.
- $W_i^0 \subseteq W_i$ is the set of initial states.
- $\delta_i \subseteq W_i \times \beta(W) \times W_i$ is a transition relation, where $\beta(W)$ denotes the set of all Boolean propositional formulae over W.
- $L_i : W_i \rightarrow 2^{O_i}$ is a labeling function that labels each state with a set of local observable events. The intuition is that $L_i(w)$ are the events that occur, or hold, in w.

Since states are labeled with sets of elements from O, we refer to $\Sigma = 2^O$ as the *alphabet* of P. While each component of P has its local observable events and its own states and transitions, these transitions depend not only on the component's current state but also on the current states of the other components. Also, as we shall now see, the labels of the components are required to agree on shared observable events.

A *configuration* of P is a tuple $c = \langle w_1, w_2, \ldots, w_n, \sigma \rangle \in W_1 \times W_2 \times \cdots \times W_n \times \Sigma$, satisfying $L_i(w_i) = \sigma \cap O_i$ for all $1 \leq i \leq n$. Thus, a configuration describes the current state of each of the components, as well as the set of observable events labeling these states. The requirement on σ implies that these labels are *consistent*, i.e., for any P_i and P_j, and for each $o \in O_i \cap O_j$, either $o \in L_i(w_i) \cap L_j(w_j)$ (in which case, $o \in \sigma$), or $o \notin L_i(w_i) \cup L_j(w_j)$ (in which case, $o \notin \sigma$). For a configuration $c = \langle w_1, w_2, \ldots, w_n, \sigma \rangle$, we term $\langle w_1, w_2, \ldots, w_n \rangle$ the *global state* of c, and we term σ the *label* of c, and denote it by $L(c)$. A configuration is *initial* if for all $1 \leq i \leq n$, we have $w_i \in W_i^0$. We use C to denote the set of all configurations of a given system P, and C_0 to denote the set of all its initial configurations. We also use $c[i]$ to refer to P_i's state in c.

For a propositional formula θ in $\mathcal{B}(W)$ and a global state $p = \langle w_1, w_2, \ldots, w_n \rangle$, we say that p *satisfies* θ if assigning **true** to states in p and **false** to states not in p makes θ true. For example, $s_1 \wedge (t_1 \vee t_2)$, with $s_1 \in W_1$ and $\{t_1, t_2\} \subseteq W_2$, is satisfied by every global state in which P_1 is in state s_1 and P_2 is in either t_1 or t_2. We shall sometimes write disjunctions as sets, so that the above formula can be written $\{s_1\} \wedge \{t_1, t_2\}$. Formulas in $\mathcal{B}(W)$ that appear in transitions are called *conditions*.

Given two configurations $c = \langle w_1, w_2, \ldots, w_n, \sigma \rangle$ and $c' = \langle w_1', w_2', \ldots, w_n', \sigma' \rangle$, we say that c' is a *successor of* c *in* P, and write $succ_P(c, c')$, if for all $1 \leq i \leq n$ there is $\langle w_i, \theta_i, w_i' \rangle \in \delta_i$ such that $\langle w_1, w_2, \ldots, w_n \rangle$ satisfies θ_i. In other words, a successor configuration is obtained by simultaneously applying to all the components a transition that is enabled in the current configuration. Note that by requiring that successors are indeed configurations, we are saying that transitions can only lead to states satisfying the consistency criterion, to the effect that they agree on the labels for shared observable events. [1]

Given a configuration c, a c-*computation* of P is an infinite sequence $\pi = c_0, c_1, \ldots$ of configurations, such that $c_0 = c$ and for all $i \geq 0$ we have $succ_P(c_i, c_{i+1})$. A *computation* of P is a c-computation for some $c \in C_0$. The computation $c_0, c_1, \ldots$ *generates* the infinite *trace* $\rho \in \Sigma^\omega$, defined by $\rho = L(c_0) \cdot L(c_1) \cdots$. We use $\mathcal{T}(P^c)$ to denote the set of all traces generated by c-computations, and the *trace set* $\mathcal{T}(P)$ of P is then defined as $\bigcup_{c \in C_0} \mathcal{T}(P^c)$. In this way, each concurrent transition system P defines a subset of Σ^ω. We say that P *accepts* a trace ρ if $\rho \in \mathcal{T}(P)$. Also, we say that P is *empty* if $\mathcal{T}(P) = \emptyset$; i.e., P has no computation, and that P is *universal* if $\mathcal{T}(P) = \Sigma^\omega$; i.e., every trace in Σ^ω is generated by some fair computation of P.

The *size* of a concurrent transition system P is the sum of the sizes of its components. Symbolically, $|P| = |P_1| + \cdots + |P_n|$. Here, for a component $P_i = \langle O_i, W_i, W_i^0, \delta_i, L_i, \alpha_i \rangle$, we define $|P_i| = |O_i| + |W_i| + |\delta_i| + |L_i| + |\alpha_i|$, where $|\delta_i| = \sum_{\langle w, \theta, w' \rangle \in \delta_i} |\theta|$, $|L_i| = |O_i| \cdot |W_i|$, and $|\alpha_i|$ is the sum of the cardinalities of the sets in α_i. Clearly, P can be stored in space $O(|P|)$.

---

[1] This requirement could obviously have been imposed implicitly in the transition relation.

When P has a single component, we say that it is a *sequential transition system*. Note that the transition relation of a sequential transition system can be really viewed as a subset of $W \times W$, and that a configuration of a sequential transition system is simply a labeled state.

Now, we introduce the definitions about the *local* and *global* treewidth, and the degree of a graph.

Definition 3.1. *The communication graph of a concurrent transition system P is a graph having one vertex for each component and an edge (v_i, v_j) if either the component for v_i and the component for v_j share observable events or if the transition relation of one of the components for v_i or v_j refer to the variables of the other.*

Definition 3.2. *The local treewidth of the concurrent transition system P is the treewidth of its communication graph.*

By the Theorem 2.2 in [16], every concurrent transition system P can be translated into a sequential transition system of size $2^{O(|P|)}$.

Definition 3.3. *The global treewidth of the concurrent transition system P is the treewidth of its equivalent sequential transition system.*

Definition 3.4. *The degree of a graph the maximum vertex degree, in other words, the maximum count of arcs connected to a single vertex in the graph.*

A graph with bounded pathwidth and bounded degree has bounded cutwidth [28]. The pathwidth bound implies many other structural restrictions [27].

Example 3.1. We construct a concurrent transition system P to encode a (ripple-carry) binary counter; it can count up to 2^n using n components. Each component P_i is used to store the i-th bit (the bit with weight 2^{i-1}), so P_1 is the least significant bit and P_n is the most significant bit. The observable events are the bit-values stored by each component, and the counter works by ripple-carry propagation.

Formally, given the number n of bits, P is $\langle \{bit_1, \ldots, bit_n\}, P_1, \ldots, P_n \rangle$, where $P_i = \langle \{bit_i\}, \{s^i_{00}, s^i_{01}, s^i_{10}\}, \{I^i\}, \delta_i, L_i \rangle$. For each state s^i_{jk}, the subscript j represent the carry status, and the subscript k represent the bit state; for example, the state s^i_{10} represents the case where the value of bit i is 0 and a carry is propagated toward bit $i+1$. I^i is an initial state, described below.

In Figure 2 we show the process P_i. The edges are labeled by the condition of the transition relation: c_{i-1} means that the carry of the process P_{i-1} is 1, and it corresponds to s^{i-1}_{10}, $\neg c_{i-1}$ means that the carry of P_i is 0 and it corresponds to $s^{i-1}_{00} \vee s^{i-1}_{01}$.

We remark that P_1 corresponds to the least significant bit of the counter, and the c_0 is always 1. We define δ_i and L_i as follows:

- $\delta_i = \{\langle s^i_{00}, \neg c_{i-1}, s^i_{00}\rangle, \langle s^i_{00}, c_{i-1}, s^i_{01}\rangle, \langle s^i_{01}, \neg c_{i-1}, s^i_{01}\rangle, \langle s^i_{01}, c_{i-1}, s^i_{10}\rangle, \langle s^i_{10}, \neg c_{i-1}, s^i_{00}\rangle, \langle s^i_{10}, c_{i-1}, s^i_{01}\rangle, \}$.
- $L_i(s^i_{00}) = L_i(s^i_{10}) = \emptyset$, $L_i(s^i_{01}) = \{bit_i\}$.

Note if we start with s^i_{00} for all states, the ripple-carry nature of the counter would take $2^n + n - 1$ cycles to flip the carry state of the most significant bit, so we initialize the counter with the binary representation of $n-1$ to ensure the carry on the most significant bit will happen after exactly 2^n cycles. The communication graph of this counter have constant pathwidth, since each component P_i interacts only with the components P_{i-1} and P_{i+1}, thus forming a path.

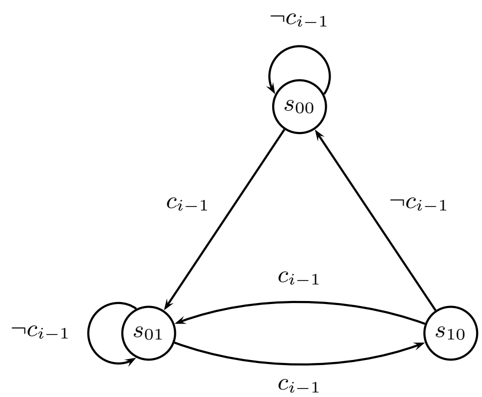

Fig. 2. A cell of the counter

In the following, we introduce the definitions for the verification problems that we consider here: model checking, containment, simulation.

The temporal logics [24] often used in the model checking are *CTL* and *LTL*, which are fragments of *CTL**. The logic *CTL** combines both branching-time and linear-time operators [13]. For the sake of simplicity, we consider LTL (Linear-Time Temporal Logic). It has three unary modal operators (X, G, and F) and one binary modal operator (U). Their meaning is: $X\phi$ is true in particular state if and only if the formula ϕ is true in the next state; $G\phi$ is true if and only ϕ is true from now on; $F\phi$ is true if ϕ will become true at some time in the future; $\phi U \psi$ is true if ψ will eventually become true and ϕ stays true until then. The semantics of LTL is based on computations of transition systems. Intuitively, $F\phi$ is true in a state of a transition system if ϕ is true in some following state, that is the transition system reaches a state in which ϕ is true. In modal logic literature, transition systems are called Kripke structure, and computations of Kripke structure are called Kripke models [10]. The model checking problem is to decide if all runs of a transition system satisfy the LTL formula. In formal verification, we encode the behavior of a system as a concurrent transition system, and a property we want to check as an LTL formula.

The problems that formalize correct trace-based and tree-based implementations of a system are *containment* and *simulation*, respectively. These problems are defined below with respect to two concurrent transition systems $P = \langle O, P_1, \ldots, P_n \rangle$ and $P' = \langle O', P'_1, \ldots, P'_m \rangle$ with $O \supseteq O'$, and with possibly different numbers of components. For technical convenience, we assume that $O = O'$. The *containment problem* for P and P' is to determine whether $\mathcal{T}(P) \subseteq \mathcal{T}(P')$. That is, whether every trace accepted by P is also accepted by P'. If $\mathcal{T}(P) \subseteq \mathcal{T}(P')$, we say that P' *contains* P and we write $P \subseteq P'$. While containment refers only to the set of computations of P and P', simulation refers also to the branching structure of the systems. Let c and c' be configurations of P and P', respectively. A relation $H \subseteq C \times C'$ is a *simulation relation* from $\langle P, c \rangle$ to $\langle P', c' \rangle$ iff the following conditions hold [21].

1. $H(c, c')$.
2. For all configurations $a \in C$ and $a' \in C'$ with $H(a, a')$, we have $L(a) = L(a')$.
3. For all configurations $a \in C$ and $a' \in C'$ with $H(a, a')$ and for every configuration $b \in C$ such that $succ_P(a, b)$, there exists a configuration $b' \in C'$ such that $succ_{P'}(a', b')$ and $H(b, b')$.

A simulation relation H is a *simulation from P to P'* iff for every $c \in C_0$ there exists $c' \in C'_0$ such that $H(c, c')$. If there exists a simulation from P to P', we say that P *simulates* P' and we write $S \preceq S'$. Intuitively, it means that the system P' has more behaviors than the system P. In fact, every tree embodied in P is also embodied in P'. The *simulation problem* is, given P and P', to determine whether $S \preceq S'$.

In this section we consider the complexity of the reachability, containment and simulation problems for concurrent transition systems, under the hypothesis of bounded treewidth both in the communication graph and in each component. The complexity of these problems has been studied in [16, 19]. We show that these problems have the same complexity of the general case, even if each component has constant size (and thus bounded treewidth and degree) and the communication graph has bounded pathwidth and degree (and hence bounded treewidth). Our results are then robust; in fact a bounded pathwidth implies many other structural restrictions [27].

In [19], the model-checking problem for temporal logics (e.g. CTL, LTL, CTL*) is shown to be PSPACE-hard, also in the reachability case. The reachability case is when the formula specifies an event that the transition system has to reach. For example in LTL, it is simply $F\psi$, where ψ is a Boolean formula. From the characteristic of the concurrent transition system used in the proof, the following theorem holds.

Theorem 3.1. *The CTL, LTL, and CTL* model checking for concurrent transition systems is* PSPACE-*hard also in the reachability case, and remains* PSPACE-*hard even if each component is fixed and the communication graph has bounded pathwidth and bounded degree.*

In [16] the simulation problem is shown to be EXPTIME-complete; from the characteristic of the concurrent transition systems used in the proof, the following theorem holds.

Theorem 3.2. *The simulation problem for concurrent transition systems is* EXPTIME-*hard, and remains* EXPTIME-*hard even if each component is fixed and the communication graph has bounded pathwidth and bounded degree.*

In [16] the containment problem is shown to be EXPSPACE-complete, but the concurrent transition systems used in the proofs have communication graphs with unbounded pathwidth and unbounded degree.

Theorem 3.3. *The containment problem for concurrent transition systems is* EXPSPACE-*hard, and remains* EXPSPACE-*hard even if each component has fixed size and the communication graph has bounded pathwidth and bounded degree.*

Proof. To prove hardness, we carry out a reduction from deterministic exponential-space-bounded Turing machines. Given a Turing machine T and input u of length n, we want to check whether T accepts the word u in space 2^n. we denote by Σ an alphabet for encoding runs of T (the alphabet Σ and the encoding are defined later). We write u' to represent the initial tape-encoding of u, i.e., if u is $u_1 u_2 \ldots u_n$, u' is $(q_0, u_1)u_2 \ldots u_n$. We then construct a transition system P_T over the alphabet $\Sigma \cup \{\$\}$, for some $\$ \notin \Sigma$, such that (i) the size of P_T is polynomial in $|T|$ and linear in n, and (ii) $\#u(\Sigma^\omega + (\Sigma^* \cdot \$^\omega)) \subseteq \mathcal{T}(P_T)$ iff T does not accept the word u. The crucial point is that using bounded concurrency, we can handle the exponential size of the tape by n components that count to 2^n.

We assume, without loss of generality, that once T reaches a final state it loops there forever. The transition system P_T accepts all traces in Σ^ω, and accepts a trace $w \cdot \$^\omega \in \Sigma^* \cdot \$^\omega$ if either

1. w is not an encoding of a prefix of a legal computation of T,
2. w is an encoding of a prefix of a legal computation of T, but, within this prefix, the computation still has not reached a final state, or
3. w is an encoding of a prefix of a legal, but rejecting, computation of T over any input.

Thus, P_T rejects a trace $w \cdot \$^\omega$ iff w encodes a prefix of a legal accepting computation of T and the computation has already reached a final state. Hence, P_T accepts all traces in $\#u(\Sigma^\omega + \Sigma^* \cdot \$^\omega)$ iff T does not accept the word u.

Now to the details of the construction. Let $T = \langle \Gamma, Q, \mapsto, q_0, F_{acc}, F_{rej} \rangle$, where Γ is the alphabet, Q is the set of states, and $\mapsto$: $(Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R\})$ is the transition function. We write $(q, a) \mapsto (q', b, \delta)$ for $\mapsto (q, a) = (q', b, \delta)$, with the meaning that when in state q and reading a in the current tape cell, T moves to state q', writes b in the current tape cell and moves its head one cell to the left or right, depending on δ. Finally, q_0 is T's initial state, $F_{acc} \subseteq Q$ is the set of final accepting states, and $F_{rej} \subseteq Q$ is the set of final rejecting states.

We encode a configuration of T by a string in $\#\Gamma^*(Q \times \Gamma)\Gamma^*$, of the form $\#\gamma_1\gamma_2 \ldots (q, \gamma_i) \ldots \gamma_{2^n})$. The meaning of this is that the j'th cell, for $1 \leq j \leq 2^n$, is labeled γ_j, T is in state q and its head points to the i'th cell.

We encode a computation of T by a sequence of configurations, which is a word over $\Sigma = \{\#\} \cup \Gamma \cup (Q \times \Gamma)$. Let $\#\sigma_1 \ldots \sigma_{2^n}\#\sigma'_1 \ldots \sigma'_{2^n}$ be two successive configurations of T in such a sequence. (Here, each σ_i is in Σ.) If we set $\sigma_0 = \sigma_{2^n+1} = \#$ and consider a triple $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle$, for $1 \leq i \leq 2^n$, it is clear that the transition function of T prescribes σ'_i. In addition, along the encoding of the entire computation, $\#$ must repeat exactly every $2^n + 1$ letters. Let $next(\sigma_{i-1}, \sigma_i, \sigma_{i+1})$ denote our expectation for σ'_i. That is, with the γ's denoting elements of Γ, we have:

- $next(\gamma_{i-1}, \gamma_i, \gamma_{i+1}) = next(\#, \gamma_i, \gamma_{i+1}) = next(\gamma_{i-1}, \gamma_i, \#) = \gamma_i$.
- $next((q, \gamma_{i-1}), \gamma_i, \gamma_{i+1}) = next((q, \gamma_{i-1}), \gamma_i, \#) = \begin{cases} \gamma_i & \text{if } (q, \gamma_{i-1}) \mapsto (q', \gamma'_{i-1}, L) \\ (q', \gamma_i) & \text{if } (q, \gamma_{i-1}) \mapsto (q', \gamma'_{i-1}, R) \end{cases}$
- $next(\gamma_{i-1}, (q, \gamma_i), \gamma_{i+1}) = next(\#, (q, \gamma_i), \gamma_{i+1}) = next(\gamma_{i-1}, (q, \gamma_i), \#) = \gamma'_i$, where $(q, \gamma_i) \mapsto (q', \gamma'_i, \delta)$. [2]
- $next(\gamma_{i-1}, \gamma_i, (q, \gamma_{i+1})) = next(\#, \gamma_i, (q, \gamma_{i+1})) = \begin{cases} \gamma_i & \text{if } (q, \gamma_{i+1}) \mapsto (q', \gamma'_{i+1}, R) \\ (q', \gamma_i) & \text{if } (q, \gamma_{i+1}) \mapsto (q', \gamma'_{i+1}, L) \end{cases}$
- $next(\sigma_{2^n}, \#, \sigma'_1) = \#$.

A necessary and sufficient condition for a trace to encode a legal computation of T on the word u is that consecutive configurations are compatible with $next$.

Now for the construction of P_T. P_T is a concurrent process with $n+1$ components. The first component, P_M is the master process that accept all the traces Σ^ω, and accept all non-accepting traces in $\Sigma^* \cdot \$^\omega$. The other components $P_1, \cdots, P_n$, are used by P_M

[2] We assume that T's head does not "fall" from the right or the left boundaries of the tape. Thus, the case where $i = 1$ and $(q, \gamma_i) \mapsto (q', \gamma'_i, L)$ and the dual case where $i = 2^n$ and $(q, \gamma_i) \mapsto (q', \gamma'_i, R)$ are not possible.

and their only task is perform the count as in Example 3.1; each of these processes is associated with a bit (P_1 with the least significant, P_n the most significant).

Let us describe the process P_M. In spirit, P_M follows the outline of the master process in [16]. In the construction of P_M, we use the following block of states G_{Σ^3}, which is used to generate sequences of triples $(\sigma_{i-1}, \sigma_i, \sigma_{i+1}) \in \Sigma^3$. G_{Σ^3} have $|\Sigma^3|$ states, each representing a triple, and labeled by the middle state. For two triples (u, u', u'') and (v, v', v''), there is an transition from the first to the second iff $u' = v$ and $u'' = v'$. P_M can either start in a clique of Σ states to generate Σ^ω, or it can start in a block of states (which we call $Init$) to generate non-accepting traces. All edges in $Init$ have condition $true$. From a state s in $Init$, we can reach a corresponding successor state, which represents the same triple as the successors of s in $Init$, in a new block of states B_s, of which every state asserts c_0 to start the count in the component P_1. In other words, $c_0 = \bigvee_{t \in B_s | s \in Init} t$. All edges into states in B_s have condition $true$, except those that go into states with label $next(s)$. As P_M progresses in B_s, the counter is counting to 2^n. The edges into the state labeled with $next(s)$ have condition $\neg s_{10}^n$, and from every state in B_s, we can move to a state which is a self loop labeled \$ with condition s_{10}^n. This asserts that the trace we are generating is not a prefix of a legal computation over T. Alternatively, P_M can also start in a clique of size $|\Sigma'|$ where $\Sigma' = \{\#\} \cup \Gamma \cup \{(Q - F_{acc}) \times \Gamma\}$, i.e., all the non-accepting symbols in Σ. Each edge in the clique have condition $true$, and each state in the clique can go to the self loop on \$ on condition $true$. This captures all the (legal or illegal) non-accepting traces on T.

It is easy to see that $|P_T|$ is polynomial in $|T|$ and linear in n. The processes $P_M, P_1, \cdots, P_n$ have constant size. P_M interacts only with P_1 and with P_n, the generic P_i interacts only with P_{i-1} and P_{i+1}: the communication graph is a ring and then it has bounded pathwidth and degree.

Now, given the word $u = u_1 u_2 \ldots u_n$, we construct P to be a concurrent transition system that generates the language $\#(q_0, u_1)u_2 \ldots u_n(\Sigma^\omega + (\Sigma^* \cdot \$^\omega))$. In fact, P can be easily taken to be a concurrent transition system with $n + 1$ components, each with $|\Sigma| + 1$ states, implemented as a shifter. In other words, the next state of component i is the current state of component $i + 1$, and component $n + 1$ can non-deterministically generate $\Sigma^\omega + (\Sigma^* \cdot \$^\omega)$. Obviously, each component is of constant size, and the concurrent transition system is of bounded pathwidth and bounded degree. It follows that T does not accept the word u iff $P \subseteq P_T$. By taking T to be an universal Turing machine, we showed that the containment problem for concurrent transition systems is EXPSPACE-hard even if each component has fixed size and the communication graph has bounded pathwidth and bounded degree.

□

References

1. A. Atserias, P.G. Kolaitis, and Moshe Y. Vardi. Constraint propagation as a proof system. In *CP 2004*, pages 77–91, 2004.
2. I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proc. 6th Conf. on Computer Aided Verification*, pages 182–193, Stanford, June 1994.

3. A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *DAC 1999*, pages 317–320, 1999.
4. P. Bjesse, J.H. Kukula, R.F. Damiano, T. Stanion, and Y. Zhu. Guiding sat diagnosis with tree decompositions. In *SAT 2003*, volume 2919 of *LNCS*, pages 315–329. Springer, 2004.
5. H.L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–23, 1993.
6. H.L. Bodlaender. Treewidth: Algorithmic techniques and results. In Igor Prívara and Peter Ruzicka, editors, *Proc. 22nd Int. Symp. MFCS, LNCS 1295*. Springer, 1997.
7. H.L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. Technical report, Universiteit Utrecht, 1998.
8. R.E. Bryant. Graph-based algorithms for boolean-function manipulation. *IEEE Trans. on Computers*, C-35(8), 1986.
9. R.E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Transaction on Computers*, 40(2):205–213, February 1991.
10. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2000.
11. R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.
12. R. Diestel. *Graph Theory*. Number 173 in Graduate Texts in Mathematics. Springer, 2000.
13. E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On the branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.
14. E.C Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proc. AAAI-90*, pages 4–9, 1990.
15. G. Gottlob and R. Pichler. Hypergraphs in model checking: Acyclicity and hypertree-width versus clique-width. *SIAM Journal on Computing*, 33(2):351–378, 2004.
16. D. Harel, O. Kupferman, and M.Y. Vardi. On the complexity of verifying concurrent transition systems. *Information and Computation*, 173(2):143–161, 2002.
17. J. Huang and A. Darwiche. Using DPLL for efficient OBDD construction. In *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2004)*, 2004.
18. M. Jurdziński. Deciding the winner in partity games is in UP $\cap$ co-UP. *Information Processing Letters*, 68:119–124, 1998.
19. O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of ACM*, 47(2):312–360, 2000.
20. K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
21. R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd Int. Joint Conf. on Artif. Int.*, pages 481–489. British Computer Society, September 1971.
22. J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *CAV'03*, volume 2725 of *LNCS*, pages 80–92. Springer, 2003.
23. M.R. Prasad, P. Chong, and K. Keutzer. Why is ATPG easy? In *Proc. of 36th ACM/IEEE conference on Design automation*, pages 22–28. ACM Press, 1999.
24. A.N. Prior. *Past, Present, and Future*. Clarendon Press, Oxford, 1967.
25. N. Robertson and P.D. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory Series B*, 35:39–61, 1983.
26. N. Robertson and P.D. Seymour. Graph minors. ii. algorithmic aspects of treewidth. *Journal of Algorithms*, 7:309–322, 1986.
27. T. Schiex. A note on CSP graph parameters. Technical Report 1999/03, INRIA, 1999.
28. D.M. Thilikos, M.J. Serna, and H.L. Bodlaender. A polynomial time algorithm for the cutwidth of bounded degree graphs with small treewidth. In *ESA'01*, volume 2161 of *LNCS*, pages 380–390. Springer, 2001.
29. D. Wang, E.M. Clarke, Y. Zhu, and J. Kukula. Using cutwidth to improve symbolic simulation and boolean satisfiability. In *IEEE International High Level Design Validation and Test Workshop (HLDVT 2001)*, page 6, 2001.