

An Efficient Algorithm for Real-Time Symbolic Model Checking*

J. Frösl, J. Gerlach[†], Th. Kropf

Institut für Rechnerentwurf und Fehlertoleranz (Prof. D. Schmid)
Universität Karlsruhe, Kaiserstraße 12, 76128 Karlsruhe, Germany
{Juergen.Froessl,Thomas.Kropf}@informatik.uni-karlsruhe.de
<http://goethe.ira.uka.de/hvg/>

Abstract: *The verification of real-time properties requires model checking techniques for quantitative temporal structures and real-time temporal logics. However, up to now, most of those problems were solved by a translation into a standard CTL model checking problem with unit-delay structures. Although usual CTL model checkers like SMV can be used then, the translation leads to large structures and CTL formulas, such that the verification requires large computation times and only small circuits can be verified. In this paper a new model checking algorithm for quantitative temporal structures and quantitative temporal logic is presented, which avoids these drawbacks. Motivated by low-level circuit verification, the implemented prover can be used for verifying general real-time systems. Although it has been proved that the complexity of the new algorithm is identical to the corresponding CTL model checking problem, the application of the new algorithms leads to significant better runtimes and larger verifiable structures. The paper presents the underlying algorithms, the complexity proof, implementational issues and concludes with experimental results, demonstrating the advantages of our approach.*

1 Introduction

Hardware verification is the formal proof that a circuit implementation behaves according to a given specification. The introduction of symbolic representation techniques for finite state machines [1] has made possible the automated verification of large circuits. This led to tools for checking the equivalence of two FSMs as well as to symbolic model checkers, where a temporal logic formula is proven valid with regard to a given symbolically represented finite-state model.

However, all those approaches are targeted at fully-synchronous circuits on gate or RT-level: implementations are treated as finite state machines, where each time step corresponds to a clock cycle. Timing effects caused by capacities and resistances of the transistor circuit may not be taken into account, leading to a coarse model of the circuit which is not detailed enough for verifying asynchronous circuits or standard-cell characterizations. The unit-delay model is caused by the underlying proof technique, since in most cases CTL (computational tree logic) model checking is performed [2]. There, a CTL temporal logic specification is checked against a unit-delay finite state structure.

If real time systems have to be treated, then quantitative reasoning about time has to be performed. In many applications (protocol verification, standard cell characterization, interface design) the specification is based on *quantitative timing* and properties like the following have to be verified: “a grant is given between 15ns and 47ns after the request signal” or “the circuit oscillates with a minimum frequency of 40MHz”. Hence we must be able to express and to efficiently verify quantitative properties.

Efficient real-time model checking requires two changes: the underlying temporal structure has to be extended to *timed transition graphs*, where transitions with arbitrary delay can be modeled, and the logic has to be augmented by quantitative temporal operators. Recently these real time logics have found considerable attention [3, 4, 5, 6, 7, 8]. However, only few proof algorithms have been proposed which are suited for efficient implementations, comparable to state-of-the-art model checkers like SMV [9]. Defining quantitative temporal operators by keeping the standard CTL unit-delay structures has been done previously [10].

However, the critical part for real time model checking is the support of timed transition graph structures. In usual model checkers, the temporal structures are symbolically treated by storing the Boolean transition relation efficiently using BDDs [11]. If time has to be considered in the transition relation, then Boolean functions cannot be used directly. Another possibility to verify real-time systems is to use the standard symbolic representation of unit delay temporal structures. Then however the non-unit delay transitions have to be expanded into sequences of intermediate states, in many cases resulting in an unacceptable increase of the model size [12].

In this paper a new algorithm for *quantitative computation tree logic* (QCTL) model checking is presented, directly supporting *quantitative temporal structures* (QTS). The key idea of our approach is to split the *timed* transition relation of the QTS into a set of BDDs, each representing the set of transitions with the same delay time. Thus, the efficiency of a symbolic BDD representation is kept although timed transitions are directly supported. A previous approach supporting timed transition systems uses a binary encoding of the transition such that the time information can be directly added to the transition relation [13]. However, this approach leads to a fur-

*. This research has been supported by Deutsche Forschungsgemeinschaft (DFG), SFB 358

[†]. Now with the Universität Tübingen, Lehrstuhl Technische Informatik, Sand 13, 72076 Tübingen, Germany

ther blow up of the BDD representing the transition relation. If g_{max} is the largest upper bound of all transition times, then additional $\lceil \log_2(g_{max}) \rceil$ variables have to be used in the transition relation, leading to smaller verifiable systems. Moreover, the number of additional variables only depends on g_{max} , thus systems with few but large time values are encoded redundantly. By splitting the timed transition relation into several BDDs, only as many BDDs as different transition times are needed. The partitioning of the relation leads to smaller BDD sizes such that larger structures are representable resulting also in many cases in faster computations.

Representing the timed transition relation as a vector of BDDs leads to new model checking algorithms for QTS and QCTL, introduced in this paper. Nevertheless, we could show that the complexity of our extended model checking algorithm is equivalent to the complexity of the algorithm for conventional temporal structures and CTL [2].

In the following, we first introduce CTL and its semantics based on temporal structures. Then timed temporal structures, QCTL and the model checking algorithm are then presented together with a complexity proof of the algorithm. The paper ends with experimental results and a conclusion.

2 CTL Model Checking

In CTL model checking [2], a model of the system to be verified is given as a temporal structure. Specifications are expressed in temporal logic formulas, which are then proven to be true with regard to the structure. A *temporal* (“Kripke”) structure TS is a labeled state-transition graph given by the tuple (AF, S, R, L) , where AF is a set of *atomic formulas*, S is a finite set of *states*, $R \subseteq S \times S$ is a *transition relation* and $L: S \rightarrow \wp(AF)$ is a function labeling every state with a set of atomic formulas. A *path* is an infinite sequence of states $s_0s_1\ldots$, such that $\forall i: (s_i, s_{i+1}) \in R$. Each temporal structure can also be represented by all its *computation trees* starting in each of its states.

The syntax of CTL includes propositional logic with additional temporal operators. The latter are used to relate states of the computation tree to each other. Each temporal operator has to be composed of a path quantifier – *universal* (“A”) or *existential* (“E”) – and one of the four operators *nexttime* (“X”), *globally* (“G”), *eventually* (“F”) or *until* (“U”). Propositional and temporal operators can be arbitrarily composed.

The semantics of a CTL formula is given with regard to a temporal structure TS and an initial state s_0 . If formula f is true in a state s_0 of a structure TS , we write $TS, s_0 \models f$. The truth value of a formula is inductively defined as follows:

If f is an atomic formula $f \in AF$ then $TS, s_0 \models f$ iff

$$f \in L(s_0).$$

$TS, s_0 \models \neg f$, iff $f \notin L(s_0)$.

$TS, s_0 \models f \wedge g$, iff $TS, s_0 \models f$ and $TS, s_0 \models g$.

$TS, s_0 \models E f$ iff there exists a path $s_0s_1\ldots$, such that

$$TS, s_1 \models f.$$

$TS, s_0 \models E f$, iff there exists a path $s_0s_1\ldots$, such that

$$\forall i, i \geq 0: TS, s_i \models f.$$

$TS, s_0 \models E(fUg)$, iff there exists a path $s_0s_1\ldots$, such that

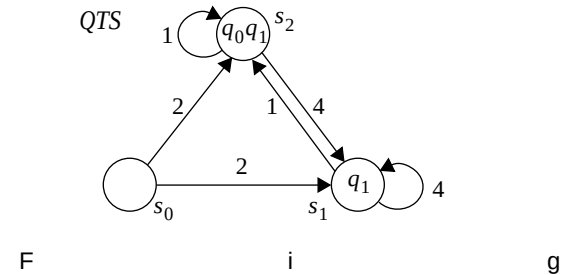
$$\exists i, i \geq 0: TS, s_i \models g \text{ and } \forall j, 0 \leq j < i: TS, s_j \models f.$$

The other temporal operators can be abbreviated as $A f \equiv \neg E \neg f$, $A f \equiv \neg E \neg f$, $E f \equiv E(t \cup f)$, $AGf \equiv \neg EF \neg f$ and $A(fUg) \equiv \neg E(\neg g U (\neg f \wedge \neg g)) \wedge AFg$.

Testing whether $TS, s_0 \models f$ holds, is performed using model checking [9]. Symbolic model checking implements this approach by representing the transition relation R of the temporal structure TS and the satisfying state sets as BDDs, the latter using the fact that a Boolean function can be interpreted as a characteristic function of a binary encoded state set.

3 Quantitative Temporal Structures

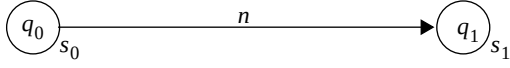
Using standard unit-delay temporal structures, it is not possible to express timed transitions explicitly. However, this is necessary if real-time systems have to be modeled. Thus we introduce quantitative temporal structures QTS that are identical to unit-delay structures TS concerning the set of atomic formulas AF , the state set S , the transition relation R , and the labeling function L . In addition we introduce a *distance function* $G: R \rightarrow (\mathbb{N} \setminus 0)$, which assigns a distance to every transition. The distance can be interpreted as a transition time (Figure 3-1).



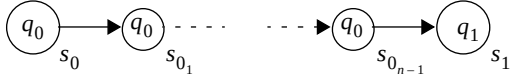
Even if we never perform this step explicitly, the semantics of a QTS transition is explained the easiest way by translating it into a behaviorally equivalent TS . For this purpose each QTS transition with distance n is expanded using $n - 1$ intermediate states, connected via unit-delay transitions. However, this can be done in two ways, that differ in the state labeling. Intermediate states are either labeled with the atomic formulas of the original QTS predecessor state (s_0 in Figure 3-2), denoting that the values do not change during the transition. A second possible interpretation “ignores” the labeling of the intermediate states such that we are not able to say anything about the values of the atomic formulas in-between (Figure 3-2). The quantitative temporal logic, introduced in the next section of this paper supports both interpretations simultaneously, leading to two variants of quantitative nexttime operators.

The semantics of a QTS is explained by the example of a CMOS-inverter. The QTS_{inv} of Figure 3-3 represents the timed behavior of the inverter using an inertial delay model. Hence, an input signal of the inverter leads to a change of the

QTS transition:



TS interpretation 1:



TS interpretation 2:

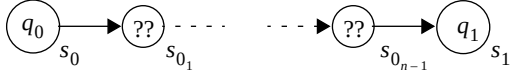


Figure 3-2: QTS interpretation

output signal only if the input signal is stable for a minimum amount of time. As temporal structures do not carry transition inputs as usual finite state machines, we have to represent input variables as additional state variables and input changes as non-deterministic state transitions.

The QTS representing the inverter has therefore four states, representing the four possible valuations of the input p and output q with binary values. Labeling a QTS-state with an atomic formula p or q means a logical value “one” of the corresponding circuit-signals. The two *stable states* of the inverter (p and q have opposite binary values) are represented by the QTS-states s_0 and s_2 having self-loops with distance one. Thus the QTS may stay in these states for an arbitrary amount of time. The two *unstable states* s_1 and s_3 (p and q have equal binary values) without self-loops have to be left after a given *delay time*.

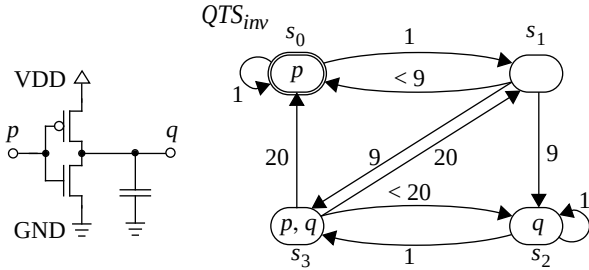


Figure 3-3: QTS representing a CMOS-inverter

If the QTS is in one of its stable states and an input change occurs, then a transition to the respective unstable state takes place. If the QTS is in one of its unstable states and the input keeps its value a state transition is performed after the upper time bound (in our example 9 and 20, respectively) of all possible transitions of this state; the corresponding transitions are labeled with this fixed distances. If a further change of the input occurs before reaching this time bound, then the QTS performs a transition back to the former stable state. The labellings <9 and <20 of the corresponding transitions in the QTS are abbreviations for the sets of all transitions with distances $1 \leq j < 9$ and $1 \leq j < 20$, respectively. It is possible and supported by our model checker to abbreviate arbitrary sets of transitions using such expressions. The transitions between s_1 and s_3 are performed if an input change occurs simultaneously to a state transition.

4 Quantitative Temporal Logics

The syntax of QCTL is a direct extension of CTL. Besides the complete CTL-syntax, we introduce the additional *quantitative operators* $EXW[n]f$, $EXS[n]f$, $EfU[n]g$ and $EG[n]f$. Their semantics is defined with regard to a quantitative temporal structure QTS as follows, where $n \in (\mathbb{N} \setminus 0)$:

$QTS, s_0 \models EXW[n]f$, iff there exists a path $s_0 s_1 \dots$ and a

$$k \in \mathbb{N}, \text{ such that } QTS, s_k \models f \text{ and } \sum_{i=0}^k G((s_i, s_{i+1})) > n.$$

$QTS, s_0 \models EXS[n]f$, iff there exists a path $s_0 s_1 \dots$ and a

$$k \in \mathbb{N}, \text{ such that}$$

$$QTS, s_k \models f \text{ and } \sum_{i=0}^{k-1} G((s_i, s_{i+1})) = n.$$

$QTS, s_0 \models EG[n]f$, iff there exists a path $s_0 s_1 \dots$ and a

$$k \in \mathbb{N}, \text{ such that } \forall i, 0 \leq i \leq k: QTS, s_i \models f \text{ and}$$

$$\sum_{i=0}^k G((s_i, s_{i+1})) > n.$$

$QTS, s_0 \models EfU[n]g$, iff there exists a path $s_0 s_1 \dots$ and a

$$k \in \mathbb{N}, \text{ such that}$$

$$QTS, s_k \models g \text{ and } \forall j, 0 \leq j < k: QTS, s_j \models f \text{ and}$$

$$\sum_{i=0}^{k-1} G((s_i, s_{i+1})) \leq n.$$

Obviously, the QCTL operators are defined analogously to their CTL counterparts, taking timing constraints into account. Note, that we have introduced a *strong* and a *weak* variant of the quantitative nexttime operator. The strong version of the quantitative nexttime operator $EXS[n]f$ holds for an actual state iff there is at least one path to an explicit QTS state in which f holds and the length of the path equals n . Thus only explicit QTS states are considered, leading to the second interpretation of Figure 3-2. The weak version $EXW[n]f$ also takes into account variable values of intermediate states, leading to the first interpretation of Figure 3-2. Thus $EXW[n]f$ also holds for each state s_i for which f holds and that is the starting point of at least one transition with a distance greater than n . This is the case because for the weak variant of the nexttime operator we assume that the labelling of s_i holds for all intermediate states of all transitions, starting from s_i .

The operators $AXW[n]$, $AXS[n]$, $AfU[n]g$, $AG[n]$, $AF[n]$ and $EF[n]$ can be derived from the explicitly introduced QCTL operators in the usual manner.

Using QCTL we can specify behavioral properties of the introduced CMOS-inverter example. The specification:

$$QTS_{inv} \models \neg EG[21](p \wedge q)$$

claims that the inverter cannot be in the unstable state in which $(p \wedge q)$ hold for 21 time units. This specification obviously holds for all states of the QTS in figure 3-3. Another example is the specification:

$$QTS_{inv}, s_0 \models EXW[10](EG(\neg p \wedge q)).$$

This formula states that there is at least one path starting in state s_0 such that $(\neg p \wedge q)$ holds after 10 time units and further holds for an arbitrary long time. This specification shows

the combination of conventional CTL-operators with the introduced quantitative operators. Using conventional CTL-Operators for the specification can be interpreted as an “untime” construction since these operators do not use the transition distances, thus abstracting from the timing information. Furthermore the composition of temporal operators allows us to specify arbitrary intervals of time. In the above example the combination of the CTL-operator $EG(\neg p \wedge q)$ and the QCTL-operator $EXW[10]$ specifies an unbounded time interval: $[10, \infty]$.

5 Symbolic Model-Checking for QTS and QCTL

In this section we show, how symbolic model checking methods for simple temporal structures TS and CTL can be adapted to quantitative temporal structures QTS and QCTL.

If n is the number of state variables, then a unit-delay temporal structure for conventional CTL model checking is symbolically given by a characteristic function of type $\mathbb{B}^{2n} \rightarrow \mathbb{B}$, representing the transition relation $R(s, s')$, where s and s' indicate the actual and next state, respectively [2]. R can be easily stored and manipulated using BDDs.

Obviously this representation cannot be directly used for the quantitative temporal structure QTS , since we have to additionally encode the distances $G : \mathbb{N}$. Instead of encoding the values directly in the transition relation [13], we represent the timed transition relation R_T which also covers the distances, by a set of “untimed” relations. Since we always have a finite number of distances, we can partition the entire set of transitions into subsets, each containing all transitions of distance i and represented by a transition relation R_T^i . Thus the overall QTS transition relation is given as an array $R_T = [R_T^1, \dots, R_T^{g_{max}}]$, ranging from one to the maximum distance occurring in a given QTS . Each of the R_T^i can be represented by a single BDD. For symbolic model checking the overall characteristic transition-function is represented as a vector of BDDs. Figure 5-1 shows the BDD representation of

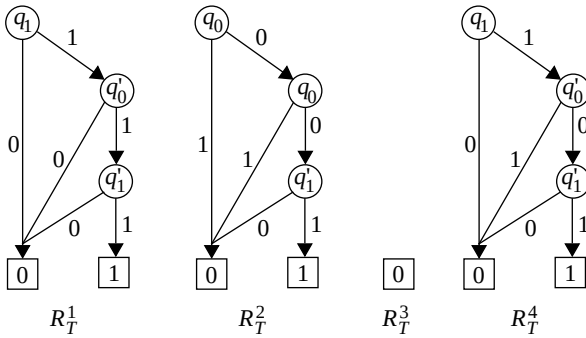


Figure 5-1: Representing the quantitative temporal structure of figure 3-1 using an array of BDDs

the QTS example given in figure 3-1. Our model checking algorithm is directly based on this partitioned transition relation.

The model checking algorithm for QTS and QCTL is closely related to the model checking algorithm for unit-delay

temporal structures and CTL, developed by Clarke, McMillan and others [2]. To check a QCTL-formula against a QTS , the model checking algorithm calculates the subset of QTS -states in which the formula holds. This is performed by scanning the syntax-tree of the given QCTL-formula bottom up while calculating the satisfying state-set of each occurring subformula. The type of a subformula can be *true*, an atomic-formula from AF , a Boolean connective $\neg f$ or $f \wedge g$, a CTL-operator EXf , $E\{fUg\}$, EGf or a QCTL-operator $EXS[n]f$, $EXW[n]f$, $E\{fU[n]g\}$ or $EG[n]f$. All other operators can be reduced to one of these basic ones. For each of these basic operators there exists a separate check function that is able to calculate the respective satisfying state set, using the partitioned transition relation, represented as a BDD.

Our model checking algorithm is not restricted to quantitative temporal operators but also allows to check pure CTL-formulas as well as formulas containing simultaneously CTL and QCTL-operators. The check functions for the Boolean connectives only operate on characteristic functions representing state sets and thus do not differ from standard CTL implementations. For checking CTL-operators we assume each transition of the QTS as of distance one. Thus the implementation of the corresponding check functions can be done using the well known fixpoint algorithms and is therefore not discussed here. The distances of the QTS -transitions are only considered in the case of QCTL-operators. The algorithms are explained in the following for $EXS[n]f$ and $EXW[n]f$.

The state set, satisfying $TS, s_0 \models EXS[n]f$ is characterized by:

$Sat(EXS[n]f) = \{s_0 \in S \mid \text{there is a path } s_0 s_1 \dots s_k \text{ such}$

that $s_k \in Sat(f)$ and $\sum_{i=1}^k G(s_i, s_{i+1}) = n\}$

The algorithm `check_EXSn` calculates the BDD, representing the state set $Sat(EXS[n]f)$. The algorithm performs a backtrace starting from the subset of states where f immediately holds until the length of the traced path equals n :

```

function Check_EXSn( $f$ :BDD,  $n$ : integer):BDD
begin
   $EX^0 := f$ 
  for  $i = 1$  to  $n$  do  $EX^i := false$  endfor
  for  $i = 0$  to  $n-1$  do
    for  $j = 1$  to  $\min\{g_{max}, n-i\}$  do
       $EX^{i+j} := EX^{i+j} \vee \exists s'(R_{[s, s']}^j \wedge EX_{[s' \rightarrow s]}^i) ^*$ 
    endfor
  endfor
  result :=  $EX^n$ 
  return result
end

```

Each of the variables EX^i carries the characteristic function of a state set. In the first line of the initialization sequence the variable EX^0 is assigned with the BDD, representing the set of states for which f immediately holds. Then each element

*. $[s \rightarrow s']$ denotes the substitution of s by s' .

of the vector $(EX^1, \dots, EX^n)$ is initialized with the BDD *false*, representing an empty set. While performing the algorithm each intermediate variable EX^i is assigned with the set of those states from which a successor state that satisfies f is reachable via a path of length i .

The kernel of the algorithm is build up with two nested loops ranging over the variables i and j . For the initial value of the outer loop ($i = 0$), the inner loop calculates those state sets, each state of which is a direct predecessor of a state for which f immediately holds. In the first run of the inner loop ($j = 1$) the subset of predecessors with distance one are determined and assigned to EX^{i+j} with $i + j = 1$. The inner loop ends if j reaches the minimum of g_{max} and $n - 1$, because we know on the one hand, that the transitions between two succeeding states can only be labeled with distances no greater than g_{max} . On the other hand it is not necessary to consider transitions with distance greater than the quantitative parameter n of $EXS[n]f$.

The calculation of the state sets EX^{i+j} is performed by evaluating the existential quantified relational product: $\exists s'(R_{[s, s']}^j \wedge EX_{[s' \rightarrow s]}^i)$. The BDD $EX_{[s' \rightarrow s]}^i$ represents the set of those states from which there exists a path of length i with a destination state for which f holds. The BDD $R_{[s, s']}^j$ represents all transitions of distance j having a destination state in the state set $EX_{[s' \rightarrow s]}^i$. The result of the relational product therefore is a BDD that represents all states with at least one transition of distance j to a successor state that itself is the starting point of a path with length i and a destination state for which f holds. Thus each evaluation of the relational product may be interpreted as a backtrace of previously determined paths of weight i by a further transition with distance j . While running the algorithm each path that ends in a state for which f holds is traced back until an original QTS state is reached such that the length of the path equals n . Each state with this property is a state for which formula $EXS[n]f$ holds.

As mentioned above the *weak next operator* $EXW[n]f$ does not require an explicit QTS state as path destination. For the example QTS shown in figure 5-2 formula $EXS[n]f$

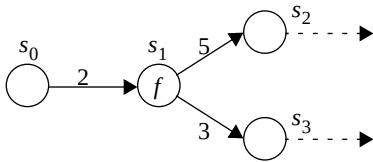


Figure 5-2: Example for strong and weak next

holds in state s_0 only for $n = 2$. In comparison the weak variant of the next operator $EXW[n]f$ holds in state s_0 for all quantitative parameters $2 \leq n < 7$, since in the case of the weak variant we assume that f holds for all (virtual) intermediate states until a successor state is reached for which f does not hold. For each state of the QTS where f holds and d_{max} is the maximum distance of all transitions starting from this state, obviously each weak next statement $EXW[n]f$ with $n \leq d_{max} - 1$ holds also. Nevertheless this semantic difference between the strong and weak variant of the QCTL-next operators only effects the initialization sequence of the corre-

sponding check-algorithm. The algorithm `check_EXWn` can therefore be obtained by simply adding

for $j = \max\{g_{max}, n\}$ to $j = 2$ do

$EX^{j-1} := \exists s'(R_{[s, s']}^j \wedge EX_{[s \rightarrow s']}^0) \vee EX^j$ endfor

at the end of the initialization sequence of the `check_EXSn` algorithm. The relational product $\exists s'(R_{[s, s']}^j \wedge EX_{[s \rightarrow s']}^0)$ calculates the set of all states for which f holds and which are the starting point of a transition with length j . For each state of this set, formula $EXW[j-1]f$ holds and is therefore assigned to EX^{j-1} . Obviously for each state in which formula $EXW[j-1]f$ holds, all formulas $EXW[n]f$ with $n < j-1$ also hold and thus we additionally have to merge the state subsets EX^{j-1} with all previously calculated subsets EX^j , by a disjunction of their BDDs. The kernel of `check_EXWn` is identical to the kernel of the check-algorithm for the strong next operator.

The check-algorithms `check_EUn` and `check_EGn` implementing the quantitative temporal operators $E\{fU[n]g\}$, and $EG[n]f$ have similar implementations and thus are not discussed here in detail.

6 Complexity of QCTL Model Checking

In the following we will show that the complexity of QCTL model checking is identical to the complexity of CTL-model checking, that is $O(|f| \cdot (|R| + |S|))$ where $|f|$ is the length of the CTL-formula we want to check and $|R| + |S|$ is the size of the temporal structure [2]. Due to the fact, that each model checking problem for QCTL can be translated to an equivalent CTL one we are therefore able to directly compare the complexities of the two problems. First we want to determine the complexity of the `check_EXW` algorithm. Let us assume, that we want to check the QCTL-formula $EXW[1]f$ and the transitions of the underlying QTS has maximum distance of $g_{max} = 1$. In this case the vector of BDDs, representing the transition-relation of the QTS has only one element and the algorithm `check_EXW` has to calculate the relational product exactly for one time. But this is exactly what the comparable CTL check-algorithm would do to solve the equivalent CTL-formula EXf [9]. The complexity to check $EXW[1]f$ for a QTS with $g_{max} = 1$ is therefore equal to $O(|f| \cdot (|R| + |S|))$. If we now allow the QTS to have a maximum distance $g_{max} \geq 1$ and the formula $EXW[n]f$ to have a quantitative parameter $n \geq 1$, the `check_EXW` algorithm have to calculate the relational product $g_{max} \cdot n$ -times in the worst case. Thus the complexity checking formula $EXW[n]f$ is $O(g_{max} \cdot n \cdot |f| \cdot (|R| + |S|))$. On the other hand if we want to check the equivalent problem with CTL, we need to encode the distances of the QTS-transitions into linear chains of states of the TS, increasing the number of states by the factor of g_{max} in the worst case. Furthermore we have to transform the QCTL-formula $EXW[n]f$ into a CTL-formula of the form $EX_1(EX_2(\dots(EX_n f)\dots))$ what increases the length of the formula by the factor of n in the worst case. As result the equivalent CTL problem also has an overall complexity of $O(g_{max} \cdot n \cdot |f| \cdot (|R| + |S|))$. Similar argumentations can be found for all other QCTL-Operators and thus we

can conclude that QCTL and CTL model checking have equal cost.

7 Results

The complexity of the QCTL model checking algorithms QCTL is identical to the complexity of the CTL model checking. In the following we want to examine whether our QCTL-model checker DELTA, based on the efficient representation of QTSs and an algorithm optimized for quantitative operators leads to any gain in the runtimes in comparison to CTL-model checking.

Judging the runtimes of the two model checking approaches by the verification of individual circuits may lead to overly optimistic or overly pessimistic results, depending on the specific circuit properties. To get results, that are independent of individual circuit properties, we checked equivalent QTSs and TSs in form of trees generated randomly with an automatic model generator. The generated trees differ in the maximum distance g_{max} of their transitions, their maximum number of branches starting in a node and the total number of nodes. The generator built up trees by arbitrarily choosing distance-values i of each transition and numbers j of branches, starting at each node. The total number of nodes and thus the size of the tree depends on the given number of atomic formulas $|AF|$. The generated QTSs are also mapped onto an equivalent TS by replacing each quantitative transition by a chain of $i - 1$ intermediate states. The diagram in figure 7-1 shows the results of checking formula EXW[12] f for various tree structures. The y-axis of the diagram gives the value V which is the quotient of the runtimes for the conventional model-checker SMV [9] and DELTA. The x-axis is labeled with the number of atomic-formulas $|AF|$ that corresponds to a number of $2^{|AF|}$ states of the tree structures. To represent the equivalent TS we need additional atomic-formulas according to the value of g_{max} . The diagram shows three curves related to structures with three different values of g_{max} . The results show that the gain in the runtimes of DELTA increases with increasing values of g_{max} and the size of the models. Even for small values of g_{max} our model checking approach leads to much better runtimes compared with conventional approaches. It is also clear that it makes no sense to use QCTL model checking to verify completely "untimed" specifications or pure functional models because of the necessary implementation overhead for quantitative structures and temporal operators.

8 Conclusions

With this paper we introduced an extended model checking algorithm, able to handle quantitative temporal structures and a corresponding quantitative temporal logic. Quantitative temporal structures are well suited to represent real-time behavior of VLSI-circuits, protocols or arbitrary real-time systems that can be represented as finite automata. The implementation of the algorithms is based on partitioning the transition relation with regard to their different distances. This partitioning allows the representation of the timed transition relations using a vector of conventional BDDs and avoids the necessity to use a possibly huge single BDD to represent the

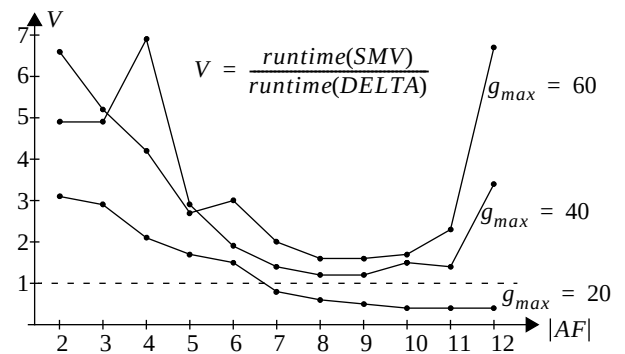


Figure 7-1: Results

transition relation in conventional model checking. Thus DELTA allows the verification of real-time systems, that cannot be represented and verified using conventional model-checkers. The above results have shown, that our implementation outperforms traditional model checking in those examples, where quantified time is used. Our current work aims at evaluating the performance gain of DELTA for the verification of MOS transistor-circuits, done up to now using standard CTL model checking [14, 12].

9 References

- [1] O. Coudert, C. Berthet, and J.C. Madre. Verification of synchronous sequential machines based on symbolic execution. In *Workshop on Automatic Verification Methods for Finite State Systems*, pages 365–373, Grenoble, June 1989.
- [2] J.R. Burch, E.M. Clarke, D.E. Long, K.L. MacMillan, and D.L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4): pages 401–424, April 1994.
- [3] A. Pnueli and E. Harel. Applications of Temporal Logic to the Specification of Real Time Systems. In M. Joseph, editor, *Formal Techniques in Real Time and Fault Tolerant Systems: Proceedings of a Symposium*, volume 331 of *Lecture Notes in Computer Science*, pages 84–98, New York, 1988. Springer-Verlag.
- [4] R. Alur and T.A. Henzinger. A Really Temporal Logic. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 164–169, New York, 1989. IEEE, IEEE Computer Society Press.
- [5] R. Koymans. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Systems*, 2: pages 255–299, 1990.
- [6] J.S. Ostroff. Deciding properties of timed transition models. *IEEE Transaction on Parallel and Distributed Systems*, 1(2):170–183, April 1990.
- [7] R. Alur, C. Courcoubetics, and D.L. Dill. Model Checking in Dense Real-time. Technical report, Stanford University, University of Crete, 1991.
- [8] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. In *7th Symposium of Logics in Computer Science*, pages 394–406, Santa-Cruz, California, June 1992. IEEE Computer Society Press.
- [9] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell Massachusetts, 1993.
- [10] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative Temporal Reasoning. *Journal of Real Time Systems*, 4:pages 331–352, 1992.
- [11] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
- [12] J. Fröhl and T. Kropf. Verifying real-time properties of MOS-transistor circuits. In *Proceeding of the European Design and Test Conference*, pages 314–319, Paris, March 1995. IEEE Computer Society Press.
- [13] S.V. Campos and E. Clarke. Real-Time Symbolic Model Checking for Discrete Time Models. In T. Rus and C. Rattray, editors, *Theories and Experiences for Real-Time System Development*, AMAST Series in Computing. World Scientific Press, AMAST Series in Computing, May 1994.
- [14] J. Fröhl and T. Kropf. A new model to uniformly represent the function and timing of MOS transistor circuits and its application to VHDL simulation. In *Proc. of the European Design Automation Conference (EDAC 94)*, pages 343–348, Paris, March 1994.