Efficient Decompositional Model Checking for Regular Timing Diagrams

Nina Amla[1], E. Allen Emerson[1], and Kedar S. Namjoshi[2]

[1] Department of Computer Sciences, University of Texas at Austin ***
{namla,emerson}@cs.utexas.edu
URL: http://www.cs.utexas.edu/users/{namla,emerson}
[2] Bell Laboratories, Lucent Technologies
kedar@research.bell-labs.com
URL: http://cm.bell-labs.com/cm/cs/who/kedar

Abstract. Timing diagrams are widely used in industrial practice to express precedence and timing relationships amongst a collection of signals. This graphical notation is often more convenient than the use of temporal logic or automata. We introduce a class of timing diagrams called *Regular Timing Diagrams (RTDs)*. RTDs have a precise syntax, and a formal semantics that is simple and corresponds to common usage. Moreover, RTDs have an inherent compositional structure, which is exploited to construct an efficient algorithm for model checking a RTD with respect to a system description. The algorithm has time complexity that is linear in the system size and a small polynomial in the representation of the diagram. The algorithm can be easily used with symbolic (BDD-based) model checkers. We illustrate the workings of our algorithm with the verification of a simple master-slave system.

1 Introduction

The design of hardware systems includes the specification of timing behavior for circuit components. In industrial practice, this behavior is most often described graphically by timing diagrams. Timing diagrams are, however, often used informally and without a precise semantics, making it difficult to utilize them for the specification and verification of correct behavior. We address this issue by introducing the class of *Regular Timing Diagrams* (RTDs); which have a simple and precise semantics and an efficient, decompositional model checking algorithm. These diagrams describe changes of signal values over a finite time period, and precedence and timing dependencies between such events, such as "signal a rises within 5 time units of signal b falling" and "signal b is low when signal a rises". The time intervals are specified by integer constants, ensuring that the diagram defines a *regular* language.

*** This work was supported in part by NSF grants CCR 941-5496, CCR 980-4736 and SRC Contract 98-DP-388.

A RTD, like the circuit it describes, may be either *asynchronous* or *synchronous*. A *synchronous* diagram includes one or more "clocks" with fixed periods; the time interval between any pair of events is determined up to the clock period. Asynchronous timing diagrams are used to specify handshaking protocols, like bus arbitration and memory access, while synchronous diagrams can specify timing requirements of clocked systems. The ordering between events is partial; such RTDs are called *ambiguous*. An unambiguous RTD has a total ordering on events (See Figure 1).

Since a RTD is defined for a finite time period, an important question that arises in defining the semantics is the manner in which an infinite computation satisfies a timing diagram? Fisler [13] considers two kinds of semantics: in the *invariant* semantics, the timing diagram must be satisfied at *every* state of a computation, while in the *basic iterative* semantics, the diagram must be satisfied iteratively, at points satisfying a precondition of the diagram. Our semantics is a reformulation of the basic iterative semantics, where we permit timing diagrams to be satisfied in an overlapping manner. For simplicity, in our current model, the precondition is a state property. In general, a precondition is a path property; it can be handled by introducing a monitor automaton for the property (see Section 2.2 for a discussion). This permits a system to satisfy diagrams that express the correctness of different aspects of its operation. For ambiguous diagrams, we further classify this semantics into a *weak* aspect, where a fresh linear ordering of the events is chosen for each satisfaction of the diagram, and a *strong* aspect, where a single linear order is chosen that applies to each satisfaction of the diagram.

The key observation that leads to efficient *model checking* [5, 22, 6] is that timing diagrams are compositional (conjunctive) in nature. This can be visualized informally as the waveforms acting independently and only interacting with other waveforms through a dependency. Rather than build the single, *monolithic NFA* (non-deterministic finite state automaton) or the temporal logic formula that corresponds to the entire diagram, we demonstrate that it is possible to decompose the diagram into properties of isolated waveforms and their interactions. This results in a conjunction of simpler properties that can be conveniently represented by a succinct $\forall$-automaton ($\forall FA$) [21, 28]. A $\forall FA$ (also known as "dual-run" or "universal" automaton) is a finite state automaton that accepts an input iff *every* run of the automaton along the input meets the acceptance criterion. $\forall FA$'s can be exponentially more succinct than *NFA*'s and naturally express properties that are conjunctive in nature.

Moreover, this conjunctivity can be exploited to verify smaller components of the timing diagram in isolation, thus avoiding the construction of the entire $\forall$-automaton. We present efficient algorithms that convert RTDs under the various semantics into $\forall FA$'s that are in the worst case of size cubic in the size of the diagram and the largest time constant represented in unary (note that the unary size is exponential in the binary size). These constants are generally performance bounds and tend to be small; thus, we feel justified in claiming polynomial complexity. The use of $\forall FA$'s permits the efficient use of the automata-theoretic language containment paradigm [29, 19, 20] to model checking. For a system M and RTD T, the verification check can be cast as $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{A}_T)$, where $\mathcal{A}_T$ is the (small, polynomial size) $\forall FA$ for the diagram T and $\mathcal{L}(X)$ denotes the language of X. This is equivalent to $\mathcal{L}(M) \ \cap \ \neg\mathcal{L}(\mathcal{A}_T) = \emptyset$. The complement language of a $\forall FA$ is accepted by a *NFA* with identical structure but complemented acceptance condition. Hence, complementation (the $\neg\mathcal{L}(\mathcal{A}_T)$ term) is trivial, and the complexity of the model checking procedure is linear in the size of the structure and the size of the $\forall FA$ $\mathcal{A}_T$. In addition, it is often possible to decompose $\mathcal{A}_T$ itself into a conjunction of smaller $\forall FA$'s, which may be checked independently with M. It is also simple to produce a description of $\neg\mathcal{L}(\mathcal{A}_T)$ that can be input to a symbolic model checker. To illustrate our method, we show how the behavior of read and write transactions that is described by RTDs can be checked against a simple master-slave memory system.

We believe that this framework permits efficient model checking of timing specifications that are used in practice. Our review of industrial data books and discussions with engineers indicate that RTDs are sufficiently expressive for most industrial verification needs. With the exception of Fisler's work [13, 14], where the model checking algorithms have high complexity, other prior work considers timing diagram models that are at most as expressive as RTDs. The algorithm is linear in the structure size, polynomial in the number of diagram points and dependencies and in the unary size of the constants. The polynomial complexity of our decompositional algorithm is a significant improvement over the earlier monolithic approaches, where the size may be exponential in the worst case. Not withstanding the Lichtenstein-Pnueli thesis [20], in practice, as one reaches the limits of applicability of symbolic model checking tools, the size of the specification is of importance. A detailed discussion of these points is in Section 5.

The rest of the paper proceeds as follows. In Section 2, we give a precise syntax and semantics for Regular Timing Diagrams. Section 3 outlines the algorithms that convert RTDs into $\forall FA$'s and the model checking procedure. Section 4 describes how the algorithms are used with with the model checker VIS [3] for the verification of a master-slave system. We conclude with a discussion of related work in Section 5.

2 Regular Timing Diagrams - Syntax and Semantics

A Regular Timing Diagram (henceforth referred to as RTD or diagram) is specified by a number of finite waveforms defined over a set of "symbolic" values SV, and timed dependencies between points on the waveforms. The set of symbolic values includes 1 (High), 0 (Low) and X (unspecified). The set SV is ordered by $\sqsubseteq$, where $a \sqsubseteq b$ iff $a = X$ or $a = b$.

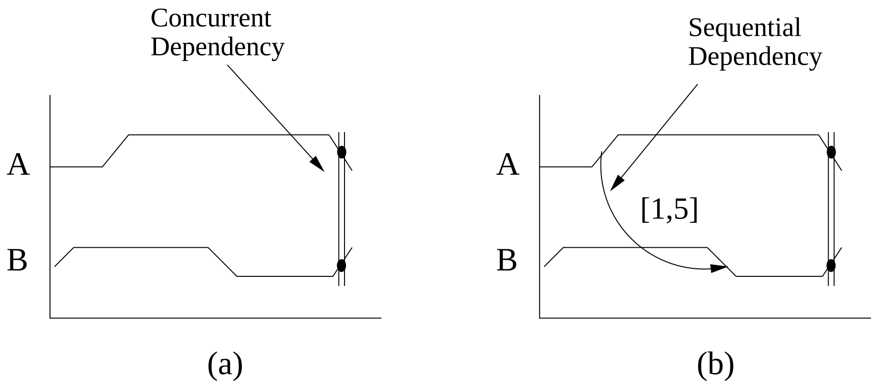

Fig. 1. (a) Ambiguous RTD (b) Unambiguous RTD

2.1 Regular Timing Diagrams: Syntax

Definition 1 (RTD) *A* RTD *is a tuple* (WF, SD, CD), *where*

- WF *is a finite set of waveforms over* SV. *Each waveform* A *of length* n *is a function* $A : [0, n) \rightarrow SV$. *A* point *of* WF *is a pair* (A, i) *where* $A \in WF$ *and* $i \in [0, length(A))$. $(A, 0)$ *is the* initial *point and* $(A, length(A) - 1)$ *is the* final *point of* A.
- SD *is the set of* sequential *dependencies on the points of* WF. *Each dependency is specified as* $(A, i) \stackrel{[a,b\rangle}{\rightarrow} (B, j)$, *where* $a \in \mathbf{N}, b \in \mathbf{N} \cup \{\infty\}$, $1 \leq a$ *and* $a \leq b$. *(The "*$\rangle$*" bracket indicates either a closed or an open interval) For convenience,* $[k, \infty)$ *is often written as* $\geq k$, $[1, k]$ *as* $\leq k$ *and* $[k, k]$ *as* $= k$.
- CD *is a collection of mutually disjoint sets of points. Each set is called a* concurrent dependency. *The set of initial and final points of the diagram form predefined concurrent dependencies.*

Definition 2 (Event) *The* events *of a RTD* (WF, SD, CD) *are defined inductively as follows, where the rules are applied in the order shown.*

1. *For every waveform* A *in* WF, $(A, 0)$ *is an event.*
2. *For an event* (A, i) *with non-X value, the first change along waveform* A *to a non-X successor value* $A(j)$ *defines* (A, j) *as an event.*
3. *If* (A, i) *is a member of a concurrent dependency that contains an event, then* (A, i) *is an event.*
4. *If* (A, i) *is an event and* $(A, i) \stackrel{=k}{\rightarrow} (B, j)$, *then* (B, j) *is an event.*

Notice that for any input string of vectors of signal values, every event has at most one position on the string. This "precise location" property of events is the key to our efficient model checking algorithm. For every event e, it is possible to construct a *DFA* we call *locator*(e) that accepts at the position on an input string where the event holds. This *DFA* essentially encodes the sequence of applications of the rules above that define the point e as an event.

A *symbolic point* of a RTD is either a concurrent dependency or a singleton set containing a point that is not in any concurrent dependency. The set of symbolic points is denoted by $\mathcal{SP}$. Informally, events in a symbolic point should occur simultaneously. The sequential dependencies of a RTD induce the following ordering relation $\prec$ on symbolic points : $p \prec q$ iff

- $(A, i) \in p$ and $(A, i+1) \in q$, for points $i, i+1$ of waveform A in WF, or
- there exist $e \in p$ and $f \in q$ such that $e \stackrel{\alpha}{\rightarrow} f$ is a sequential dependency.

The RTD syntax allows several definitions that run counter to intuition. For instance, dependencies may be cyclically related, or it may be possible that the location of a dependency is imprecise due to the presence of X (undetermined) parts of a waveform. These cases are ruled out by giving a notion of "well-formed" RTDs, which is defined below.

Definition 3 (Well-formed RTD) *A RTD is* well-formed *iff (i) every point of the RTD is an event and (ii) the transitive closure of $\prec$ ($\prec^+$) is not reflexive.*

The RTD in Figure 2 can be expressed notationally as follows.

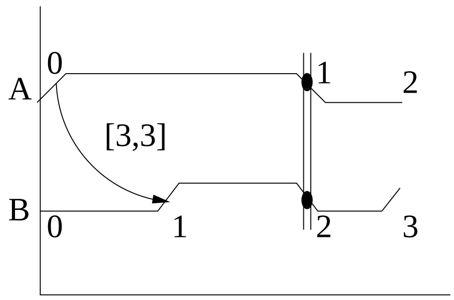

Fig. 2. Example : An Asynchronous RTD

$$
\begin{aligned}
WF &: \{A, B\} \\
A &: 0 \mapsto 1, 1 \mapsto 0, 2 \mapsto 0 \\
B &: 0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 0, 3 \mapsto 1 \\
SD &: \{(A, 0) \stackrel{[3,3]}{\rightarrow} (B, 1)\} \\
CD &: \{\{(A, 0), (B, 0)\}, \{(A, 1), (B, 2)\}, \{(A, 2), (B, 3)\}\}
\end{aligned}
$$

There are four symbolic points in this RTD : $\{(B, 1)\}$ together with the elements of CD.

2.2 Regular Timing Diagrams: Semantics

The semantics of a RTD is a set of infinite computations over *states*; each state is a vector indexed by the waveforms of the timing diagram. The set of states is denoted by Σ. The operator $\sqsubseteq$ defined earlier is extended to states as follows: $u \sqsubseteq w$ iff for each i, $u(i) \sqsubseteq w(i)$. A computation of the system to be verified consists of an infinite sequence of states from Σ. Since the syntax of a RTD describes only finite sequences of events, a key question is the appropriate extension to infinite computations.

The predefined initial concurrent dependency defines a state predicate that is thought of as the begin-condition of the finite sequence of events described by the RTD syntax; the final concurrent dependency (end-condition) is a path predicate. For example, the begin-condition for the RTD in Figure 2 is $\langle A = 1, B = 0\rangle$ and the end-condition is $\langle A = 0, B = 1\rangle$. As another example, if the diagram represents the behavior for a "memory-read" transaction, the begin- and end- conditions indicate the states that define the extent of this transaction. Clearly, this diagram should be checked only on the finite sub-computation that starts at a state satisfying the begin-condition and ends with a state satisfying the end-condition. It is sometimes necessary to make the begin-condition a *path* predicate; the path predicate identifies a sequence of states that indicate the start of a transaction. Such a path predicate can be handled in our current framework by constructing a "monitor" automaton that emits a signal whenever the path condition is satisfied; the presence of this signal, which is a state predicate, can be used as the begin-condition of the RTD.

One may thus consider an infinite sequence to satisfy a timing diagram iff the dependencies of the diagram are satisfied in each finite sub-sequence defined by the begin- and end- conditions. This statement, though, is still open to many interpretations, some of which are considered below. We first define what it means for a finite sequence of states to satisfy a timing diagram. Recall that the relation $\prec^+$ partially orders the set of symbolic points, $\mathcal{SP}$.

Definition 4 (Assignment) *An* assignment π *for a string* σ *of length* n *is a function* $\pi : \mathcal{SP} \to [0, n)$, *that is strictly monotonic w.r.t.* $\prec$ *(*$p \prec q$ *implies* $\pi(p) < \pi(q)$*) and maps the initial point of* $\mathcal{SP}$ *to* 0.

Two assignments $\pi : \mathcal{SP} \to [0, n)$ and $\xi : \mathcal{SP} \to [0, m)$ are *equivalent* iff they order symbolic points identically w.r.t. $<$ and $=$. Any assignment π induces the function $\hat{\pi} : \mathcal{P} \to [0, n)$ which maps a point (A, i) to k iff the (unique, by definition) symbolic point that includes (A, i) is mapped to k by π. From the definition of π, it follows that all points in a concurrent dependency are assigned a common position.

Definition 5 (RTD satisfaction) *A RTD* $T = (WF, SD, CD)$ *is satisfied by a finite sequence* y *over* Σ^+ *w.r.t. an assignment* $\pi : \mathcal{SP} \to [0, |y|)$ *(written as* $y \models_\pi T$*) iff the following conditions hold.*

- *Point consistency: For every point* (A, i), *if* $\hat{\pi}((A, i)) = k$, *then* $A(i) \sqsubseteq y_k(A)$. *Note that* $y_j(A)$ *denotes the value of waveform* A *at time* j *in* y.

- *Waveform consistency: Let* $\hat{\pi}((A,i)) = k$ *and* $\hat{\pi}((A,i+1)) = l$. *For every* $j \in [k,l)$, $A(i) \sqsubseteq y_j(A)$.
- *Dependency consistency: For every sequential dependency* $e \stackrel{[a,b\rangle}{\rightarrow} f$, $(\hat{\pi}(f) - \hat{\pi}(e)) \in [a,b\rangle$.

For many systems, it is the case that the begin- condition for the timing diagram does not recur before the end- condition holds. For such systems, we may consider the following semantics. System computations may be described by the expression $(\Delta^+ \vee (\#\Delta^+\$))^\omega$, where # and \$ are special vectors of Σ representing the satisfaction of the begin- and end- conditions respectively and $\Delta = \Sigma \backslash \{\#, \$\}$. The sequence of the form $\#\Delta^+\$$ is called a *transaction*.

Definition 6 (Weak Iterative Semantics) *An infinite sequence z satisfies a RTD T under the* weak iterative semantics *(written as* $z \models_w T$*) iff for every transaction* $\#y\$$ *on z, there exists an assignment* π *for which* $\#y\$ \models_\pi T$.

Definition 7 (Strong Iterative Semantics) *An infinite sequence z satisfies a RTD T under the* strong iterative semantics *(written as* $z \models_s T$*) iff there exists an assignment* ξ *such that for every transaction* $\#y\$$ *of z, there is an equivalent assignment* π *such that* $\#y\$ \models_\pi T$.

A notable class of systems where the assumption of non-overlapping transactions does not hold is those that involve some measure of pipelining. We may then consider the following generalization of the weak iterative semantics.

Definition 8 (Overlapping Semantics) *An infinite sequence z satisfies a RTD T under the* overlapping semantics *(written as* $z \models_o T$*) iff wherever # holds along z, there exists y such that* $\#y\$$ *is a prefix of the suffix computation from that point and for some assignment* π, $\#y\$ \models_\pi T$.

For the rest of the paper, we consider only the weak and strong iterative semantics in detail; the algorithm for the overlapping semantics is a slight modification of that for the weak iterative semantics and has the same complexity. We consider now an alternative formulation of Definition 5, which forms the basis for the decompositional algorithms for model checking. If $\#y\$$ satisfies the timing diagram, each event, by Definition 2 may be located precisely on the sequence. The key observation is that, since each dependency consists of precisely located events, it can be checked independently of the others. Let *pt* be the partial function that defines the location of events on a finite sequence.

Theorem 1 *For a RTD* $T = (WF, SD, CD)$, *and any finite transaction* $z = \#y\$$, *there exists an assignment* π *such that* $z \models_\pi T$ *iff each of the following conditions holds:*

- *Every event of T can be located on z and has a value consistent with that in T; i.e., pt is total, and if* $pt(z,(A,i)) = k$ *then* $A(i) \sqsubseteq z_k(A)$.
- *Let* $pt(z,(A,i)) = k$ *and* $pt(z,(A,i+1)) = l$. *For every j in* $[k,l)$, $A(i) \sqsubseteq z_j(A)$.

– *For each sequential dependency* $e \overset{[a,b\rangle}{\rightarrow} f$, $(pt(z,f) - pt(z,e)) \in [a,b\rangle$.
– *For each pair of events* e, f *in a concurrent dependency,* $pt(z,e) = pt(z,f)$.

Notice that the theorem essentially transforms the existential ($\exists$) condition of Definitions 6 through 8 into a universal ($\forall$) condition; this forms the basis for the decompositional check.

3 Decompositional Model Checking

Theorem 1 is fundamental to decomposing RTDs into a conjunction of properties of individual waveforms and ordering or timing restrictions on their interactions, which is the key to efficient model checking. In this section, we provide algorithms that translate a RTD under both strong and weak iterative semantics into a $\forall$*FA*. The basic iterative and overlapping semantics can be similarly handled. For clarity, we often describe the *NFA* for the complement language instead of the $\forall$*FA*.

Definition 9 ($\forall$*FA*) *A* $\forall$*FA on infinite strings* $\mathcal{A} = (\Sigma, Q, \delta, q_0, \Phi)$ *has a finite input alphabet* Σ*, finite state set* Q*, transition relation* $\delta \subseteq Q \times \Sigma \times Q$*, start state* q_0 *and acceptance condition* Φ*.*

A *run* r of $\mathcal{A}$ on input x in Σ^ω is an infinite sequence of states of $\mathcal{A}$, where r_0 is an initial state, and for each i, $(r_i, x_i, r_{i+1}) \in \delta$. $\mathcal{A}$ accepts x by $\forall$-acceptance according to Φ iff *every* run r on x satisfies Φ. We define $\mathcal{L}(\mathcal{A})$ to be the set of strings accepted by $\mathcal{A}$; $\mathcal{L}_{NFA}$ by a $\exists$-acceptance and $\mathcal{L}_{NFA}$ by $\forall$-acceptance. In this paper, we consider Φ to be a Büchi acceptance condition. For any $\forall$*FA* $\mathcal{A}$, let $\overline{\mathcal{A}}$ be the *NFA* with the same transition relation but complemented acceptance condition $\neg\Phi$.

Theorem 2 *([21, 28]) For any* $\forall$*FA* $\mathcal{A}$*,* $\neg\mathcal{L}_{\forall FA}(\mathcal{A}) = \mathcal{L}_{NFA}(\overline{\mathcal{A}})$.

3.1 RTDs under the weak iterative semantics

We describe here the *NFA* that accepts the complement of the weak-iterative language of a RTD $T = (WF, SD, CD)$. First, construct finite string automata for each waveform and dependency as follows:

– Waveform : If $(A, i+1)$ is defined in terms of (A, i), then *locator*$((A,i))$ is extended to ensure that the signal values up to the change of value that defines $(A, i+1)$ are above $A(i)$ in $\sqsubseteq$ order. Otherwise, *locator*$((A,i))$ is used to determine that the value at the position where (A,i) holds is above $A(i)$ in $\sqsubseteq$ order.
– Sequential dependency : For a sequential dependency $e \overset{[a,b\rangle}{\rightarrow} f$, the automaton is a parallel composition of *locator*(e) and *locator*(f) that accepts iff the time between the acceptance of the locator *DFA*'s is within $[a,b\rangle$.

- Concurrent dependency : The $\forall FA$ for a concurrent dependency C checks that for a fixed event e in C and every other event f in C, $locator(e)$ and $locator(f)$ accept at the same position on the input sequence.

The ω-*NFA* for the complement language operates as follows on an infinite input sequence : it nondeterministically "chooses" a transaction $\#y\$$ on the input, "chooses" which waveform or dependency fails to hold of the transaction, and accepts if the automaton for that entity (defined as given above) rejects. Notice that each automaton defined above is either a *DFA* or a $\forall FA$, both of which can be trivially complemented. The $\forall FA$ obtained from this *NFA* by complementing the acceptance condition defines the language of the RTD under the weak iterative semantics. Denote this $\forall FA$ by $\mathcal{A}_T$. For the diagram $T = (WF, SD, CD)$, let L be the size in unary of the largest constant in SD. Define $|T| = \#points + |SD| + |CD|$. The size of $\mathcal{A}_T$ is cubic in $|T|$ and L.

Theorem 3 (Correctness) *For any RTD T and $x \in \Sigma^\omega$, $x \models_w T$ iff $x \in L(\mathcal{A}_T)$. The size of $\mathcal{A}_T$ is polynomial in $|T|$ and the unary length of the largest constant in T.*

3.2 RTDs under the strong iterative semantics

Under the strong iterative semantics, every transaction on an input computation has to satisfy the RTD w.r.t. a single event ordering. The *NFA* for the complemented language accepts a computation iff

- Some transaction violates a waveform or dependency constraint. This is checked by the automaton defined for the weak-iterative semantics. Or,
- There is a transaction and a pair of events that occur in a different order from that in the first transaction. This is done by an automaton that "chooses" a pair of events unordered by $\prec^+$, executes the locator *DFA*'s for these events in parallel on the first transaction to determine their order, then "chooses" a subsequent transaction and executes the locator *DFA*'s of the same events on that transaction to determine the new order, and accepts if the orders differ.

Let $\mathcal{A}_T$ denote the $\forall FA$ obtained from this *NFA* by complementing the acceptance condition. The size of $\mathcal{A}_T$ is cubic in $|T|$ and L for the first case; for the second, it is quadratic in $|T|$ and L with a multiplicative factor of the number of event pairs (which is bounded by $(\#points)^2$).

Theorem 4 (Correctness) *For any RTD T and $x \in \Sigma^\omega$, $x \models_s T$ iff $x \in L(\mathcal{A}_T)$. The size of the $\forall FA$ $\mathcal{A}_T$ is polynomial in $|T|$ and L.*

3.3 Model Checking

The translation of a RTD to a small $\forall FA$ implies that the language containment approach to model checking based on [29] gives an efficient algorithm. We need to check that $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{A}_T)$, where M is the system to be verified and $\mathcal{A}_T$ is the $\forall FA$ for the RTD T. This is equivalent to $\mathcal{L}(M) \cap \neg\mathcal{L}(\mathcal{A}_T) = \emptyset$. Complementation (the $\neg\mathcal{L}(\mathcal{A}_T)$ term) is trivial for a $\forall FA$; the complemented automaton (a *NFA*) has the same structure but complemented acceptance condition. Hence, the emptiness check can be done in time linear in the size of the structure and a small polynomial in the size of T. The space complexity, by the results of [25], is logarithmic in the sizes of both M and T.

Theorem 5 *For a transition system M and a RTD T, the time complexity of model checking is linear in the size of M and a small polynomial in the size of T and the unary size of the largest constant in T.*

An alternative way of utilizing the $\forall FA$ construction is to note that, for the weak iterative semantics, the automaton essentially defines a language $(\Delta^+ \vee \#(\bigwedge_i L_i)\$)^\omega$, where the L_i's represent the languages of the dependencies. The lemma below shows that the ω-repetition distributes over the $\bigwedge_i$ in the following sense.

Lemma 1 *For finite-string languages L_i $(i \in [0, n))$ which are subsets of Δ^+, $(\Delta^+ \vee \#(\bigwedge_i L_i)\$)^\omega = \bigwedge_i(\Delta^+ \vee \#L_i\$)^\omega$.*

By this lemma, one can construct smaller ω-automata, one for each dependency, and check that the language of each has an empty intersection with $\mathcal{L}(M)$. This is often more efficient than the combined check, and may lead to quicker detection of any errors. We refer to this as the "decompositional" approach.

4 Applications

We demonstrate the use of these algorithms in the verification of a master-slave memory system using the model checker VIS, which is based on the automata-theoretic (language containment) approach to model checking. This example is small and is intended only as an illustration of how our algorithms may be used.

In the master-slave system (Figure 3), the master issues a read or a write instruction by asserting the corresponding line, and the slaves respond by accessing memory and performing the operation. The master chooses the instruction, puts the address on the address bus and then asserts the *req* signal. The slave whose tag matches the address awakens, services the request, then asserts the *ack* line on completion. Upon receiving the *ack* signal the master resets the *req* signal, causing the slave to reset the *ack* signal. Finally, the master resets the address and data buses. The memory read (Figure 4) and write cycles are specified as RTDs (interpreted under the weak iterative semantics).

The master-slave system was simplified by abstracting away some inessential details. First, the address bus was simplified to the tag of the slaves. Since VIS does not allow variables to be both input and output, the bidirectional data bus is represented as two 1-bit boolean variables, *Idata* and *Odata* that denote the input and output data buses respectively. The begin-condition for the read RTD

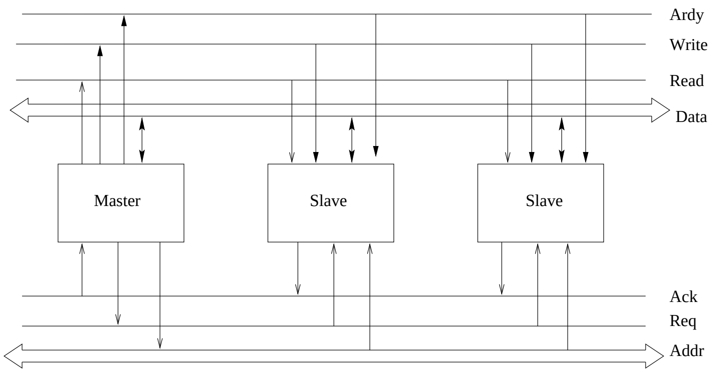

Fig. 3. Master-Slave Architecture

is the state that has *Ardy*, *Idata*, *Req*, *Ack* and *Write* being assigned 0 (low), the value of the address bus *Addr* is unknown and the *Read* signal is high. The end-condition for the read RTD is the state following the diagram where all the signals are low and *Addr* is X. Observe that during a read transaction the *write* signal remains unchanged and vice versa. This is a way of expressing negation in a timing diagram.

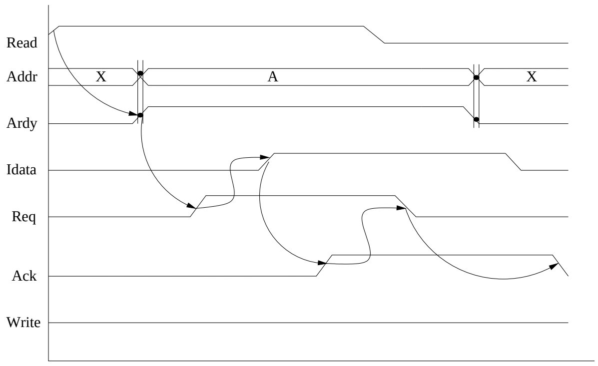

Fig. 4. RTD for the memory read cycle

The simplified master-slave system is represented in Verilog. For both RTDs, we created (as Verilog modules) both the complement of the $\forall$*FA* and the complement *NFA* for individual dependencies and waveforms.

The language emptiness check passed for both the ambiguous read and write RTD translations. The results in Table 1 show that the decompositional procedure is indeed feasible and that the size of the system to be verified together

Table 1. Verification statistics

Design under verification	Number of BDD variables	Reachable state space	Number of BDD nodes
Master-Slave Design	61	109	275
Master-Slave and 1 waveform module	70	110	330
Master-Slave and 1 dependency module	70	390	532
Master-Slave and all read RTD modules	169	17796	2019

with a single dependency automaton may not be significantly larger, in terms of BDD variables, than the system itself.

5 Conclusions and Related Work

Several researchers have investigated timing diagrams and their use in automated verification. Boriello [2] proposes an approach to formalizing timing diagrams. Timing diagrams are described informally as regular expressions but no specific details or translation algorithms are given. Many other researchers [1, 26, 23, 4] have formalized timing diagrams and translated them to other formalisms (interval logics, trigger graphs etc.). Cerny et al. present a procedure [18] for verifying whether the finite behavior of a set of action diagrams (timing diagrams) is consistent; [17] uses constraint logic programming to check if a system satisfies finite action diagram specifications. Formal notions of timing diagrams have also proved to be useful in test generation and logic synthesis (cf. [27, 15, 12]).

Fisler [13, 14] proposes a timing diagram syntax and semantics that allows non-regular languages, and finds that these languages occur at all levels of the Chomsky hierarchy. The paper [14] provides a decision procedure that determines whether a regular language is contained in an unambiguous timing diagram language, and [13] provides an algorithm that translates a certain class of timing diagrams into CTL [5]. A key difference with our work is that this algorithm is restricted to a subset of unambiguous timing diagrams under the basic invariant semantics (our algorithms under both iterative and invariant semantics are defined for all types of diagrams). The regular containment procedure [14] has a high complexity (in PSPACE), while our algorithms have *polynomial* time complexity in the diagram size.

An important contribution in this area is the work done by Damm and colleagues at the University of Oldenburg on Symbolic Timing Diagrams (STD's) [9, 24, 8, 16, 7]. STD's may be compiled into first-order temporal logic formulae which are then used for model checking. STD's are extended in [11, 10] to

RTSTD's (Real-time STD's), where a translation into a timed propositional temporal logic TPTL is provided. Both these research efforts consider infinite languages and ambiguity. A key difference with our work lies in the fact that their translation is monolithic, in the sense that all dependencies are considered together; this can result in an exponential blowup in the size of the resulting formulae when the diagram is highly ambiguous. While it is possible to model check the first order temporal logic presented in [9, 10], the procedure is not very efficient.

This paper presents "regular" timing diagrams (RTDs), which have a simple syntax and precise, simple semantics that closely corresponds to common usage. From our discussions with engineers, we are led to believe that RTDs are expressive enough to represent many timing diagrams that arise in practice. As mentioned earlier, the algorithms proposed in this paper can also be used with synchronous RTDs.

Noteworthy contributions of this paper include polynomial time, decompositional algorithms for model checking timing diagram specifications, which are based on a decomposition of the RTD semantics into properties of each waveform and the way they interact. Such decompositions may also provide a way of composing RTDs and thereby building new RTDs hierarchically. Our algorithms generate a $\forall FA$ (NFA) corresponding to the RTD (the negation of the RTD). We can choose to use either the $\forall FA$ (by splitting it into smaller $\forall FA$'s) or its complement NFA in verifying that a system satisfies a RTD. These algorithms are a significant improvement over the earlier possibly exponential, monolithic translations. We have shown how our algorithms may be used in conjunction with a symbolic model checker such as VIS, to verify systems with specifications formulated as RTDs.

We are currently working on a tool that implements these translation and verification algorithms. We also intend to test the efficiency of our algorithms on industrial strength examples.

Acknowledgments We would like to thank Bob Kurshan, Kathi Fisler and Steve Keckler for helpful discussions and insightful comments.

References

1. C. Antoine and B. Le Goff. Timing Diagrams for Writing and Checking Logical and Behavioral Properties of Integrated Systems. In *Correct Hardware Design Methodologies*. Elsevier Sciences Publishers, 1992.
2. G. Borriello. Formalized Timing Diagrams. In *EDAC92*. IEEE Comput. Soc. Press, 1992.
3. R. Brayton, G. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, S. Sarwary, T. Shiple, G. Swamy, and T. Villa. VIS. In *FMCAD*, 1996.
4. V. Cingel. A Graph-based Method for Timing Diagrams Representation and Verification. In *Correct Hardware Design and Verification Methods*. Springer Verlag, 1993.

5. E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In *Workshop on Logics of Programs*, volume 131. Springer Verlag, 1981.
6. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems using Temporal Logic. *ACM Transactions on Programming Languages and Systems*, 8(2), 1986.
7. W. Damm and J. Helbig. Linking Visual Formalisms: A Compositional proof System for Statecharts cased on Symbolic Timing Diagrams. In E. R. Olderog, editor, *Programming Concepts, Methods and Calculi*. Elsevier Science B.V. (North Holland), 1994.
8. W. Damm, H. Hunger, P. Kelb, and R. Schlör. Using Graphical Specification Languages and Symbolic Model Checking in the Verification of a Production Cell. In C. Lewerentz and T. Lindner, editors, *Formal Development of Reactive Systems. Case Study Production Cell, LNCS 891*. Springer Verlag, 1994.
9. W. Damm, B. Josko, and Rainer Schlör. Specification and Verification of VHDL-based System-level Hardware Designs. In Egon Borger, editor, *Specification and Validation Methods*. Oxford University Press, 1994.
10. K. Feyerabend. Real-time Symbolic Timing Diagrams. Technical report, Department of Computer Science, Oldenburg University, September 1994.
11. K. Feyerabend and B. Josko. A Visual Formalism for Real Time Requirement Specifications. In *AMAST Workshop on Real-time systems and Concurrent and Distributed Software*. Springer Verlag, 1997.
12. K. Feyerabend and R. Schlör. Hardware synthesis from requirement specifications. In *EURO-DAC'96 with EURO-VHDL'96*. IEEE Computer Society Press, September 1996.
13. K. Fisler. *A Unified Approach to Hardware Verification Through a Heterogeneous Logic of Design Diagrams*. PhD thesis, Computer Science Department, Indiana University, August 1996.
14. K. Fisler. Containment of Regular Languages in Non-Regular Timing Diagrams Languages is Decidable. In *CAV*. Springer Verlag, 1997.
15. W. Grass, C. Grobe, S. Lenk, W. Tiedemann, C.D. Kloos, A. Marin, and T. Robles. Transformation of Timing Diagram Specifications into VHDL Code. In *Conference on Hardware Description Languages*, 1995.
16. J. Helbig, R. Schlör, W. Damm, G. Doehmen, and P. Kelb. VHDL/S - Integrating Statecharts, Timing diagrams, and VHDL. *Microprocessing and Microprogramming*, 1996.
17. F. Jin and E. Cerny. Verification of Real-Time Controllers against Timing Diagram Specifications using Constraint Logic Programming. In *IFIP EuroMICRO*, 1998.
18. K. Khordoc and E. Cerny. Semantics and Verification of Timing Diagrams with Linear Timing Constraints. *ACM Transactions on Design Automation of Electronic Systems*, 3(1), 1998.
19. R.P. Kurshan. *Computer-aided verification of coordinating processes: the Automata-theoretic approach*. Princeton University Press, 1994.
20. O. Lichtenstein and A. Pnueli. Checking that Finite State Concurrent Programs satisfy their Linear Specifications. In *POPL*, 1985.
21. Z. Manna and A. Pnueli. Specification and Verification of Concurrent Programs by $\forall$-Automata. In *POPL*, 1987.
22. J.P. Queille and J. Sifakis. Specification and Verification of Concurrent Systems in CESAR. In *Proc. of the 5th International Symposium on Programming*, volume 137 of *LNCS*, 1982.

23. Y.S.. Ramakrishna, P.M. Melliar-Smith, L.E. Moser, L.K. Dillon, and G. Kutty. Really Visual Temporal Reasoning. In *Real-Time Systems Symposium*. IEEE Publishers, 1993.
24. R. Schlör. A Prover for VHDL-based Hardware Design. In *Conference on Hardware Description Languages*, 1995.
25. A.P. Sistla, M. Vardi, and P. Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *TCS*, 49, 1987.
26. E.M. Thurner. Proving System Properties by means of Trigger-Graph and Petri Nets. In *EUROCAST*. Springer Verlag, 1996.
27. W.D. Tiedemann. Bus Protocol Conversion: from Timing Diagrams to State Machines. In *EUROCAST*. Springer Verlag, 1992.
28. M. Vardi. Verification of Concurrent Programs. In *POPL*, 1987.
29. M. Vardi and P.Wolper. An Automata-Theoretic approach to Automatic Program Verification. In *LICS*, 1986.