Containment of Regular Languages in Non-Regular Timing Diagram Languages is Decidable

Kathi Fisler*

Department of Computer Science
Rice University
6100 S. Main, MS 132
Houston, TX 77005-1892
kfisler@cs.rice.edu

Abstract. Parametric timing constraints are expressed naturally in timing diagram logics. Algorithmic verification of parametrically constrained timing properties is a difficult problem, known to be undecidable in most general cases. This paper establishes that a class of parametrically constrained timing properties can be verified algorithmically against finite-state systems; alternatively stated, containment by a regular language is shown decidable for a class of language properties (regular and non-regular) expressible in our timing diagram logic.

1 Introduction

Timing diagrams provide a linear-time temporal logic that is well suited to expressing timing constraints. When variables can appear in timing constraints, the resulting timing diagram logic can express context-free and context-sensitive language properties. Algorithmic verification of such non-regular properties against finite-state system specifications is generally known to be undecidable. Timing diagrams, however, can only express non-regular languages with particular structural characteristics. This raises the question of whether the class of timing diagram languages is amenable to algorithmic verification. This paper establishes that containment by a regular language is decidable for a class of timing diagram languages, implying that certain non-regular language properties expressible as timing diagrams can be algorithmically tested against finite-state systems.

Timing diagrams have been used formally in a variety of hardware reasoning tasks. Brzozowski, Gahlinger, and Mavaddat provided algorithms for testing consistency and satisfiability of timing specifications given as timing diagrams in the context of interfacing components [3]; similar efforts have been undertaken by Cerny and Khordoc [4]. Several researchers have proposed using algebras of timing diagrams annotated with various programming language constructs for

* This research was conducted while the author was a graduate student at Indiana University, with financial support from AT&T Bell Laboratories under the PhD Fellowship Program.

the behavioral specification of designs [6, 9, 10]. Algorithmic verification has been applied to requirements expressed as timing diagrams by translating the diagrams into existing formalisms such as VHDL [11] and timed automata [2]. Although some of these efforts support quantitative timing constraints (those using numeric constants), none support *parametric* timing constraints (those allowing variables over numerals).

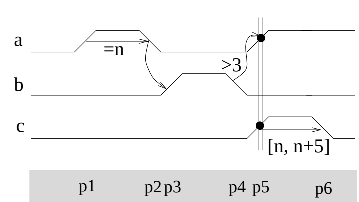
Alur, Henzinger, and Vardi have studied parametric timing constraints for real-time systems [1]. They defined a theory of parametric timed automata with multiple clocks for tracking parametric values and established that language emptiness is decidable when one clock is constrained by parameters, undecidable when three or more clocks are constrained by parameters, and an open problem when two clocks are so constrained. The timing diagrams considered in this work can correspond to problems that would require multiple parameterized clocks. We do not solve the general question of decidability of language emptiness for two clock systems. Rather, this work shows that timing diagrams correspond to a class of parameterized timing problems potentially requiring multiple clocks for which containment by a regular language is decidable.

2 Timing Diagrams and Their Languages

This work uses a formal logic of timing diagrams (called TDL) developed as part of our study of diagrammatic representations as formal specification languages for design and verification [5]. Starting from fairly common timing diagram notations, we define timing diagram semantics relative to formal languages. TDL is expressively incomparable to existing temporal logics such as LTL. In particular, LTL cannot express parametric timing constraints, while TDL cannot express properties such as $\mathsf{F}p$ and $\mathsf{G}(p \to q)$; reasons for this are given in Section 2.2.

2.1 Syntax

Waveforms depict transitions between low and high voltage levels; timing diagrams depict relations over the levels and transitions appearing within a set of waveforms. TDL supports two relations: synchronization and temporal ordering. Temporal ordering relationships may be constrained with discrete-time lower and upper bounds. We allow these bounds to contain variables that range over the natural numbers; these variables introduce parametric timing constraints, as shown in the following example.



Vertical parallel lines indicate synchronization of the levels or transitions on which the circles appear. The term *event* refers to a transition on a waveform or a synchronization. The arrows indicate temporal ordering, while the annotations on the arrows indicate the lower and upper bounds on the time passing between the related events: annotation "=n" (shorthand for [n,n]) indicates that the lower and upper bounds are the same, while annotation "> 3" (shorthand for [3, ∞]), indicates a lower bound of 3 but no upper bound. Unannotated arrows have time bound [1, ∞] by default. *Valid bound expressions* consist of natural numbers, variables, and arbitrary addition and subtraction expressions over them, as well as the symbol ∞. The labels in the shaded area are for explanatory purposes only and are not part of the timing diagram.

Formally, a timing diagram contains three components: (1) an ordered sequence P of *time points*, which are abstract moments of time at which events occur; (2) a function N from names to *waveforms*, defined as functions from P to voltage level designators of type H (for high level), L (for low level), F (for falling transition), and R (for rising transition); and (3) a ternary relation O on $N \times P \times B$ capturing temporal ordering, synchronization and time bounds, where B consists of time bound expressions of the form $[l, u]$, such that l is a non-∞ valid bound expression and u is any valid bound expression.

As an example, we construct tuple $\langle P, N, O\rangle$ to capture the above timing diagram. P contains time points $\{p_1, p_2, p_3, p_4, p_5, p_6\}$, as shown in the shaded area. N is constructed from the levels on each waveform at each time point.

$$\begin{aligned} N = \{&(a, \{(p_1, R), (p_2, F), (p_3, L), (p_4, L), (p_5, R), (p_6, H)\}),\\ &(b, \{(p_1, L), (p_2, L), (p_3, R), (p_4, F), (p_5, L), (p_6, L)\}),\\ &(c, \{(p_1, L), (p_2, L), (p_3, L), (p_4, L), (p_5, R), (p_6, F)\}) \end{aligned}$$

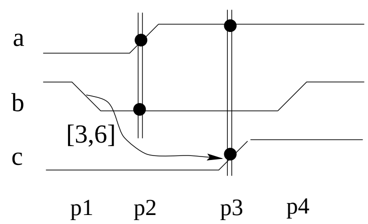
Relation O contains five elements, the first four corresponding to the arrows and the fifth to the synchronization line:

$$\begin{aligned} O = \{&((a, p_1), (a, p_2), [n, n]), ((a, p_2), (b, p_3), [1, \infty]), ((b, p_4), (a, p_5), [3, \infty]),\\ &((c, p_5), (c, p_6), [n, n + 5]), ((a, p_5), (c, p_5), [0, 0])\} \end{aligned}$$

Although we have provided only an example here, the process of representing a timing diagram in this tuple form can be formalized as a straightforward parsing procedure. In the remainder of the paper, the term "timing diagram" refers to the tuple form, rather than the original picture.

2.2 Semantics

Timing diagrams are modeled by (finite or infinite) words over an alphabet containing all possible assignments of boolean values to the names labeling waveforms. Intuitively, a word models a timing diagram when the transition patterns in the diagram reflect the changes in values assigned to names in the word. One difficulty in formalizing timing diagrams is that their intended meanings differ widely between contexts and among users. Rather than fix one interpretation, TDL has been parameterized to allow user customization of the semantics. These parameters are illustrated using the following timing diagram and word.



a	0	0	0	1	1	0	1	1	0	1	1	1
b	1	0	0	0	1	0	0	1	0	0	0	1
c	0	0	0	0	0	1	0	1	0	0	0	1
	0	1	2	3	4	5	6	7	8	9	10	11
	p_1		p_2	p_4	p_3							
					p_1	p_2	p_3, p_4					
								p_1	p_2		p_3, p_4	

The word is presented in tabular form: the rows are labeled with waveform names and the columns with the indices into the word. The three lines containing time points beneath the table indicate three separate assignments of indices to time points, as explained in the example below.

We allow timing diagrams to specify assume-guarantee relationships between their events. One parameter indicates those time points that comprise the "assume" portion. For purposes of this example, take p_1 as the only such time point. Semantically, we will begin to match the word against the timing diagram starting at the first index that matches the events in the assumed portion, in this case index 0. Next, we walk the word looking for the smallest indices that match any events that can immediately follow the falling transition on b, in this case the first synchronization line (p_2).

A second parameter is motivated by the attempt to assign an index to the first synchronization line. The line itself requires that b be low when a rises; however, is b required to be low *until* a rises? The desirability of the "until" interpretation is heavily context-dependent. Therefore, the user may indicate segments of waveforms that should be matched exactly within words. Such segments are called *fixed-level constraints* and have the form (n, p, p'), where n is a waveform name and p and p' are time points such that the waveform corresponding to n shows a single voltage level between p and p'. For this example, assume we have only one such constraint: (a, p_2, p_3). Then the first synchronization line is matched at index 2, while the rising transition on b is matched at index 3 and the second synchronization line at index 4: this assignment appears in the first line below the table. Note that despite the appearance that the rising transition on b must follow the one on c, the semantics does not enforce this since the relationship was not indicated explicitly using an arrow.

TDL requires timing diagrams to be matched repeatedly in a word. Two notions of repetition appear useful: one in which the next repetition starts after the previous one has been completed, and the other in which a new repetition starts in any index satisfying the "assume" portion of the diagram. We therefore define two semantic relationships: $\models_{\mathrm{Iter}}$ (for iterative) and $\models_{\mathrm{Inv}}$ (for invariant), respectively. Under the invariant semantics, the next match would begin at index 4, while under the iterative semantics, the next match would begin at index 5; these matches are shown in the next two lines beneath the table. Note that the match attempted from index 4 is not valid since the assigned indices violate the bounds on the arrow. This word would therefore model the diagram under the iterative semantics, but not under the invariant semantics.

This example provides insight into the expressive nature of TDL. For example, TDL can express context-sensitive languages such as $(a^n b^n c^n)^*$ (taking a^n to mean $\langle a = 1, b = 0, c = 0\rangle$, with similar interpretations of b^n and c^n). TDL can not express more general non-regular languages such as "the number of a's equals the number of b's". Relative to temporal logic, TDL can not express LTL formula $\mathsf{F}p$ because there is no way to stop the repeated searches at a particular point. TDL can not express $\mathsf{G}(p \to q)$ because it is not possible to make disjunctive statements within a TDL timing diagram. In separate work, we are investigating calculi over timing diagrams that would relax these restrictions [5].

Due to space constraints, only the invariant semantics is defined in the remainder of this section. The iterative semantics is defined formally in [5]. Tuples $\langle T, S, X\rangle$ capture a timing diagram, its set of assumed time points, and its fixed-level constraints; the term "timing diagram" is henceforth overloaded to also refer to one of these tuples.

Given a timing diagram, we can derive a partial order on its time points from the ordering of events within individual waveforms and the temporal ordering relationships. For a time point p, the *enabling points* of p are those time points p' such that p' precedes p in this partial order. If the order is total, the timing diagram is called *temporally unambiguous*.

An *index assignment* is a partial function I from time points to natural numbers indexing a word; if I is not total, the time points on which it is defined must form a prefix of the partial order on time points. Defn. 1 details the conditions an index assignment must meet in order to satisfy the requirements of a prefix of time points. Intuitively, the assigned indices must satisfy the events occurring at each time point while respecting the fixed-level constraints and time bounds on all events defined within those time points. For index i into a word W, $W_i(n)$ denotes the value of W on the signal named n at index i.

Definition 1 Let $D = \langle\langle P, N, O\rangle, S, X\rangle$ be a timing diagram. Let U be a prefix of the partial order of P and let I be an index assignment for U over a word W. *I satisfies the constraints of U relative to D* iff

1. For each time point $p \in U$, $I(p)$ *satisfies* p; *i.e.*, for every waveform name n in N, $W_{I(p)}(n) = 0$ (resp. 1) and $W_{I(p)+1}(n) = 1$ (resp. 0) if n has a rising (resp. falling) transition at p and $W_{I(p)}(n) = 0$ (resp. 1) if n has a low (resp. high) level at p and there is a synchronization line through n at p.
2. Each variable appearing in a time-bound expression in O can be replaced by a natural number such that for each $((a, p), (a', p'), [l, u])$ in O, if p and p' are both in U, then $l \leq I(p') - I(p) \leq u$.
3. For each fixed-level constraint (n, p, p') in X, if p and p' are both in U, then for each index i such that $I(p) < i < I(p')$, $W_i(n) = 0$ (resp. 1) if n has a low (resp. high) level between p and p'.

It follows from part 2 of this definition that variables in timing constraints are treated as existentially quantified within a single index assignment. However, the instantiations of variables need not be consistent across the many index assignments produced while repeatedly matching the diagram against the word.

Starting from a given index, there are potentially many index assignments satisfying Defn. 1. The semantic definitions must rely on a particular such index assignment. We have chosen to use the one that assigns to each time point the smallest index that satisfies it while respecting the partial order among the time points; this index assignment will be called *minimal.*

Definition 2 Let $D = \langle\langle P, N, O\rangle, S, X\rangle$ be a timing diagram, W be a word, i be an index into W, and U be a prefix of the partial order on P. Index assignment I is *minimal* for D, W, i, and U iff I satisfies the constraints of U relative to D and for each time point $p \in U$, $I(p) \geq i$ and $I(p)$ is the smallest index of W that satisfies p and is larger than all indices assigned to the enabling points of p.

The semantics places one other restriction on index assignments in addition to minimality: they must be defined on as large a prefix of the time point partial order as possible. An index assignment is called *fully minimal* if it is minimal and cannot be extended to a minimal index assignment for the same timing diagram, word, and starting index, but with a larger prefix of time points. Fully minimal index assignments are unique for temporally unambiguous timing diagrams.

Given a timing diagram $\langle T, S, X\rangle$ and a word W, the invariant semantics starts from each index of W in turn and locates the fully minimal index assignment. If that assignment is defined for all time points in S, the semantics requires that it be defined for all time points in T. If this property fails for some index of W, then W fails to model $\langle T, S, X\rangle$.

Definition 3 Let $D = \langle T, S, X\rangle$ be a timing diagram and let W be a word. D *invariantly describes* W (denoted $W \models_{\mathrm{Inv}} D$) iff for all indices i into W, whenever the fully minimal index assignment for D, W, i, and S is defined for all time points in S, it is defined for all time points in T. Given a language L, D *invariantly describes* L (denoted $L \models_{\mathrm{Inv}} D$) iff for every word $W \in L$, $W \models_{\mathrm{Inv}} D$. The set of all words that invariantly describe D is denoted $\mathcal{L}(D)^{\mathrm{Inv}}$.

3 Decidability

This section establishes that containment of a regular language in a temporally unambiguous timing diagram language is decidable; we refer to the general problem of containment by a regular language as the *regular containment problem.* Formally, a *timing diagram language* is any language that can model a timing diagram under either the iterative or the invariant semantics. The proof for the invariant semantics is discussed in detail; the proof for the iterative semantics is outlined. Our decision procedures are based on a correspondence between temporally unambiguous timing diagram languages and the languages accepted by deterministic, two-way, 1-counter machines (1-2DCM). Given a temporally unambiguous timing diagram, our algorithm creates one 1-2DCM if the diagram is interpreted invariantly, and two such machines if the diagram is interpreted iteratively. Then, given a finite automaton, we determine whether the language of the automaton models the timing diagram by computing relations on the states of the automaton using the counter machine(s). The algorithms discussed here operate on finite-state automata accepting by final state. With a slight modification, they can be tailored to operate on Büchi automata; the Büchi construction is not presented here for lack of space.

A 1-2DCM has a finite-state control, a two-way, read-only head over a finite, bounded-length input tape, and one counter which can store any natural number. Transitions are based on the current state, the letter being read, and whether the counter contains zero; the transition indicates a next state, which direction, if any, to move the input head, and whether to increment, decrement, or hold the value of the counter. The following formal definition is adapted from [7].

Definition 4

1. A two-way 1-counter machine M is a tuple $\langle K, \Sigma, \lhd, \rhd, \delta, q_0, F\rangle$ where K, Σ, $\lhd$, $\rhd$, q_0, and F are the states, inputs, left and right endmarkers, initial state, and accepting states, respectively. δ is a mapping from $K \times (\Sigma \cup \{\lhd, \rhd\}) \times \{0,1\}$ into $K \times \{-1,0,1\} \times \{-1,0,1\}$.
2. A *configuration* of M on an input $\lhd x \rhd$, for $x \in \Sigma^*$, is given by a tuple $(q, \lhd x \rhd, i, c)$ denoting the fact that M is in state q with the input head reading the i^{th} symbol of $\lhd x \rhd$, and value c is in the counter.
3. Relation $\Rightarrow$ is defined between configurations as follows: $(q, \lhd x \rhd, i, c) \Rightarrow (p, \lhd x \rhd, i+d, c+d_c)$ if a is the i^{th} symbol of $\lhd x \rhd$ and $\delta(q, a, \lambda(c))$ contains (p, d, d_c) where $\lambda(c) = 0$ if $c = 0$ and $\lambda(c) = 1$ if $c \neq 0$. $\Rightarrow^*$ denotes the transitive closure of $\Rightarrow$.
4. A string $x \in \Sigma^*$ is *accepted* by M if $(q_0, \lhd x \rhd, 1, 0) \Rightarrow^* (q, \lhd x \rhd, i, c)$ for some $q \in F$, $1 \leq i \leq |\lhd x \rhd|$, and non-negative integer c. The set of strings accepted by M form the *language* of M and will be denoted $\mathcal{L}(M)$.

Languages accepted by 1-2DCM can be characterized by the number of times the counter changes between incrementing and decrementing while reading the input tape. Denoting this parameter by r, the following results about 1-2DCM(r) are due to Ibarra *et al.* [8]:

Theorem 1 (Ibarra *et al.*, 1993)

- *The emptiness problem for 1-2DCM(r) is decidable for every $r \geq 1$.*
- *$\bigcup_r$ 1-2DCM(r) is effectively closed under complementation, intersection, and union.*
- *The containment and equivalence problems for $\bigcup_r$ 1-2DCM(r) are decidable.*

This result indicates that we can use 1-2DCM in decision procedures if we can bound the number of counter reversals made while processing any input. Such machines are called *reversal bounded.*

Given a timing diagram $\langle T, S, X\rangle$, we construct a 1-2DCM called M_{FAIL} that accepts exactly those words for which the fully minimal index assignment starting from the first position of the word is defined for all the time points in S, but undefined for some time point in T. Intuitively, the machine walks the word from the starting index, looking for indices to assign to time points; the transitions used to search for each time point's index also check for violations of the fixed-level constraints. If an index satisfying the time point is found, the machine tests any time-bounds on temporal ordering arrows whose target is at the recently matched time point. Constraints are tested one at a time by repeatedly sweeping over the input word. M_{FAIL} moves into an accepting state as soon as a violation of either the fixed-level constraints or the time-bound requirements is found. If indices corresponding to all of the time points are located, M_{FAIL} moves into a looping state from which nothing is accepted.

By construction, M_{FAIL} rejects certain words that do not model $\langle T, S, X\rangle$. In particular, this applies to words for which the fully minimal index assignment from position 0 is defined for all time points in T, but the fully minimal index assignment from some later starting position is undefined for some time point in T. The restriction of M_{FAIL} to accepting only words failing on the index assignment from the first position is important for the decidability of the problem. Based on the syntax of T, we can bound the number of counter reversals required for a 1-2DCM to test the time bound constraints of T over an arbitrary word; this follows from results in [5]. Therefore, we can bound the number of counter reversals required in a test of some fixed number of index assignment searches over a given timing diagram. As there is no fixed upper bound for the number of searches required in testing an entire word, our algorithm must perform searches only in finite increments. The restriction to single searches is sufficient for either semantics, since it follows from the definitions that any word failing to model a timing diagram has a suffix accepted by M_{FAIL}.

The decision procedure for the invariant semantics is fairly simple: the language generated by DFA A is contained in the language of timing diagram $\langle T, S, X\rangle$ iff no reachable state of A can generate a word accepted by M_{FAIL}. Formally, let $A = \langle Q, \Sigma, \delta, q_0, Q_F\rangle$ be a DFA that accepts by final state. Notation $A|_{(q,q')}$ denotes A modified to have q as the only start state and q' as the only final state, where q and q' are both in Q. We define a set Avoid containing exactly those states of A from which a word in M_{FAIL} is accepted as follows:

$$\mathsf{Avoid} =_{\text{def}} \{q \in Q \mid \exists q_f \in Q_F \;:\; \mathcal{L}(A|_{(q,q_f)}) \cap \mathcal{L}(M_{\text{FAIL}}) \neq \emptyset\}$$

Theorem 2 *Let $A = \langle Q, \Sigma, \delta, q_0, Q_F\rangle$ be a finite automaton and $D = \langle T, S, X\rangle$ be a timing diagram. Let M_{FAIL} be the machine constructed for D as described above. Let* Avoid *be computed as defined above for A. $\mathcal{L}(A) \subseteq \mathcal{L}(D)^{\text{Inv}}$ iff no state reachable from q_0 under δ is in* Avoid*.*

Proof

1. Assume $\mathcal{L}(A) \subseteq \mathcal{L}(D)^{\text{Inv}}$. From Defn. 3, every suffix of every word accepted by A invariantly describes D. Hence there does not exist a state of A reach-

able from q_0 from which a word can be generated such that the fully minimal index assignment for S is defined for all time points in S, but for which the fully minimal index assignment for all time points in T is undefined for some time point. By construction, any states in Avoid cannot be reachable from q_0, so Theorem 2 holds.

2. Assume $\mathcal{L}(A) \nsubseteq \mathcal{L}(D)^{\mathrm{Inv}}$. From Defn. 3, there must exist some index i into W from which the fully minimal index assignment for S is defined for all time points in S, but undefined for some time point in T. By construction, the suffix of W starting at index i is accepted by M_{FAIL}. Let q_i be the state A was in when it processed index i; q_i must be in Avoid by construction. The existence of W proves that q_i is reachable from q_0, so containment is correctly determined to fail. □

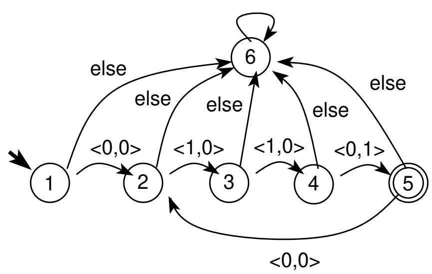
Theorem 2 suggests a decision procedure. Both M_{FAIL} and the set Avoid are constructible. Any DFA can be converted into a 1-2DCM by augmenting the transitions of the DFA with a counter whose value is never changed. The intersection of two reversal-bounded, 1-2DCMs is effectively constructible and is also a reversal-bounded, 1-2DCM by Theorem 1. The emptiness test on the intersection machine is decidable by Theorem 1.

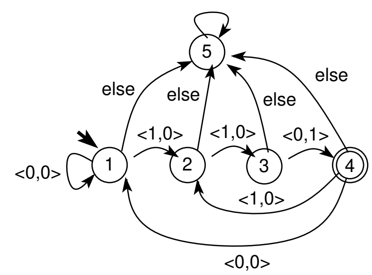
We outline the decision procedure for the iterative semantics via an example. The set Avoid is also used in deciding containment under the iterative semantics. However, the procedure for the iterative semantics is harder because the existence of a reachable state of A from which a word in M_{FAIL} is generated is not sufficient. The iterative semantics only starts a test in an index if the previous test ended in the preceding index. The decision procedure therefore needs a way to track which states in Avoid can serve as starting states for tests under the iterative semantics. We accomplish this by constructing a second 1-2DCM called M_{EXACT} which accepts those words meeting two conditions: (1) the fully minimal index assignment constructed from the first position in the word is either undefined for some time point in S or it is defined for all time points in T; (2) the last index of the word is the index in which the subsequent index assignment search must start. The justification for the first position search is the same as it was for M_{FAIL}. The restriction on the final index of the word is necessary so that we can compose words accepted by M_{EXACT} into longer words iteratively modeled by the timing diagram.

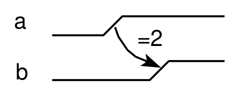
Given DFA A that accepts by final state, we use M_{EXACT} to compute a binary relation $\mathsf{TD\text{-}Reach}$ on the states of A such that $(q, q') \in \mathsf{TD\text{-}Reach}$ iff $A|_{(q,q')}$ accepts some word in M_{EXACT}. Formally,

$$\mathsf{TD\text{-}Reach} =_{\mathrm{def}} \{(q, q') \mid \mathcal{L}(A|_{(q,q')}) \cap \mathcal{L}(M_{\mathrm{EXACT}}) \neq \emptyset\}$$

Intuitively, once $\mathsf{TD\text{-}Reach}$ and Avoid are computed, we decide iterative description by checking whether there exists a state q of A such that q is in Avoid and (q_0, q) is in the transitive closure of $\mathsf{TD\text{-}Reach}$. The language of A models $\langle T, S, X\rangle$ iteratively iff no such state exists. As examples, consider the following two finite automata A_1 (left) and A_2 (right) and timing diagram T with the time point of the rising transition on a in S and nothing in X. Let $D = \langle T, S, X \rangle$.







The language of D is

$$\left(\left(\langle 0,1\rangle + \langle 1,0\rangle + \langle 1,1\rangle\right)^* \langle 0,0\rangle^+ \langle 1,0\rangle \langle 1,0\rangle \left(\langle 0,1\rangle + \langle 1,1\rangle\right)\right)^*$$

where each pair $\langle x, y\rangle$ denotes that $a = x$ and $b = y$. Furthermore, note that

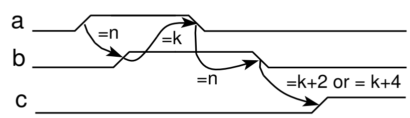
$$\mathcal{L}(A_1) = \left(\langle 0,0\rangle \langle 1,0\rangle \langle 1,0\rangle \langle 0,1\rangle\right)^* \qquad \mathcal{L}(A_2) = \left(\langle 0,0\rangle^* \langle 1,0\rangle \langle 1,0\rangle \langle 0,1\rangle\right)^*$$

The language of A_1 is contained in the language of D. By definition, $\mathsf{Avoid} = \{2,3,4\}$ and $\mathsf{TD-Reach} = \{(1,5),(5,5),(1,6),(2,6),(3,6),(4,6),(5,6),(6,6)\}$. The only states accessible from the start state (1) in the closure of $\mathsf{TD\text{-}Reach}$ are 5 and 6. As neither of these is in Avoid, $\mathcal{L}(A_1) \subseteq \mathcal{L}(D)$. The language of A_2, however, is not contained in the language of D. By definition, $\mathsf{Avoid} = \{1,2,3,4\}$ and $\mathsf{TD-Reach} = \{(1,4),(4,4),(1,5),(2,5),(3,5),(4,5),(5,5)\}$. The start state is listed in Avoid, indicating that $\mathcal{L}(A_2) \not\subseteq \mathcal{L}(D)$.

4 Future Work

Although we have proven that the regular containment problem is decidable for temporally unambiguous timing diagram languages, we do not yet have an efficient decision procedure. Testing containment of a regular language in a 1-2DCM language lies in PSPACE. Our decidability proof relied on quadratically many such tests with respect to the number of states in the automaton for the regular language. Methods for reducing the number of containment tests required remains an important problem for future work, as does the empirical analysis of the overall procedure. In addition, our current restriction to temporally unambiguous timing diagrams can likely be removed by altering the construction algorithms for M_{FAIL} and M_{EXACT}.

We are also interested in the general problem of language containment for timing diagram languages. Although the results presented here could be used to test containment of a regular-language timing diagram in an arbitrary timing diagram, they do not decide the general problem since arbitrary 1-2DCM have an unbounded number of states. It is possible to build a 1-2DCM that accepts the entire language, rather than just a single pass, of any timing diagram. Intuitively, the machine is a modification of M_{EXACT} that begins a subsequent check in the appropriate index into the word after the previous check has been completed and allows the previous pass check to walk off the end of a word. Unfortunately, this machine is not guaranteed to have bounded counter reversals. Results governing the undecidability of language containment for 1-2DCM with unbounded reversals are also inapplicable because there exist 1-counter non-reversal-bounded languages that cannot be captured in any timing diagram. Consider the "timing diagram"



which is not well-formed in our syntax due to the disjunction in the time-bound expression. Although no well-formed timing diagram has exactly the same language as this one, a 1-2DCM could be constructed to accept the language of this diagram using techniques similar to those used in constructing M_{FAIL} and M_{EXACT}. Therefore, the undecidability of language containment for 1-counter non-reversal bounded languages does not prove the undecidability of containment for timing diagram languages. We are investigating this general problem as well as possible syntactic characterizations of general timing diagram language containment.

5 Conclusions

This paper has established that the regular containment problem is decidable for any temporally unambiguous timing diagram language, regardless of where it falls in the Chomsky hierarchy. This result holds for both finite regular languages and infinite regular languages accepted by Büchi automata. We see two main implications of this result. First, there is the practical implication that certain non-regular language properties are amenable to algorithmic verification against finite-state systems; an implementation of these ideas would extend the scope of algorithmic verification, as existing logics such as LTL express only regular language properties.

The second implication is more foundational in nature. The formal methods community has largely treated diagrams as interface tools. Diagrammatic representations certainly have advantages in this regard, as evidenced by their popularity. Unfortunately, the interface approach to diagrams has lead us to focus more on how diagrams can be used to represent existing sentential logics, rather than on the computational models suggested by the diagrams in their own right. This work establishes that the structure naturally imposed by diagrammatic representations also offers advantages on a theoretical level, thus making diagrammatic representations worthy of investigation outside of the realm of interface design.

Acknowledgements

The author thanks Moshe Vardi and the anonymous reviewers for their helpful comments on this paper.

References

1. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *Proc. of the 25th ACM Symposium on the Theory of Computing*, pages 592–601, 1993.
2. Bachi Berkane, Simona Gandrabur, and Eduard Cerny. Timing diagrams: semantics and timing analysis. LASSO Laboratory, University of Montreal, 1996.
3. J.A. Brzozowski, T. Gahlinger, and F. Mavaddat. Consistency and satisfiability of waveform timing specifications. *Networks*, **21**:91–107, 1991.
4. E. Cerny and K. Khordoc. Interface specifications with conjunctive timing constraints: realizability and compatibility. In *Second AMAST Workshop on Real-Time Systems*, June 1995.
5. Kathryn Fisler. *A Unified Approach to Hardware Verification Through a Heterogeneous Logic of Design Diagrams.* PhD thesis, Indiana University, August 1996.
6. Werner Grass, *et al.* Transformation of timing diagram specifications into VHDL code. In *Proc. of Computer Hardware Description Languages and Their Applications*, pages 659–668, August 1995.
7. Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, January 1978.
8. Oscar H. Ibarra, Tao Jiang, Nicholas Tran, and Hui Wang. New decidability results concerning two-way counter machines and applications. In *Proc. of the 20th International Colloquium on Automata, Languages, and Programming*, 1993. Lecture Notes in Computer Science 700.
9. K. Khordoc, M. Dufresne, E. Cerny, P. A. Babkine, and A. Silburt. Integrating behavior and timing in executable specifications. In *Proc. of Computer Hardware Description Languages and their Applications*, pages 385–402, April 1993.
10. Philippe Mooeschler, Hans Peter Amann, and Pausto Pellandini. High-level modeling using extended timing diagrams. In *Proc. of the European Design Automation Conference*, pages 494–499, 1993.
11. Rainer Schlör. A prover for VHDL-based hardware design. In *Proc. of Computer Hardware Description Languages and Their Applications*, August 1995.