

Efficient Approximate Symbolic Reachability of Discrete-timed Digital Circuits

Sudeep Juvekar

Ankur Taly

Varun Kanade

Supratik Chakraborty

Abstract—Timed reachability analysis of gate-level circuits is important in several applications. In this paper, we present techniques for efficient approximate symbolic reachability of circuits assuming discrete delays of gates. We exploit local interactions among gates to develop a highly scalable algorithm for over-approximating the set of timed reachable states. We present a scheme of successive over-approximations and provide a probabilistic analysis to prove that these approximations converge quickly on an average. We argue that this scheme corresponds to successively extracting trees in the underlying factor graph representing the interaction of gates. We report experimental results on a set of benchmarks that demonstrate the effectiveness of our approach.

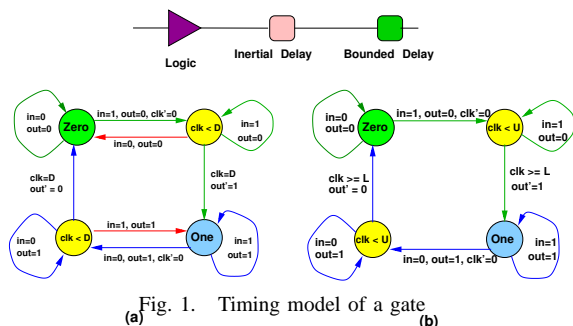
I. INTRODUCTION

Timing analysis of digital circuits has several applications in CAD. Given a delay model for each gate and a specification of how the primary inputs change with time, a core timing analysis problem is to compute the timed reachable state space of the circuit starting from a known initial state. While special algorithms have been developed [1] to solve specific timing analysis problems (e.g. critical path delay estimation) for circuits with simple gate delay models, it is difficult to extend these algorithms to more complex gate delay models or to other timing analysis problems. Timed reachability analysis offers a generic approach for solving a large class of such problems. For example, by using suitable monitors or observers, timed reachability analysis can be used to detect short glitches on gate outputs, to check time separations of transitions, etc. With realistic gate delay models, each gate requires several state variables to model its behaviour. Consequently, the total number of state variables required to model the timed behaviour of the complete circuit exceeds manageable limits even for medium-sized circuits with several hundred to a thousand gates. This poses serious challenges in using conventional reachability analysis techniques (explicit or symbolic) for exhaustively searching the reachable timed state space. In this

paper, we propose a practical and scalable approach to solve this problem, by exploiting locality of interaction between gates and wires, and drawing inspiration from summary computation techniques based on factor graphs.

Timed simulation and time symbolic simulation [2] have been used in the past to analyze the timed behaviour of circuits. With complex gate delay models, the total number of possibilities to simulate explodes very soon. In time symbolic simulation, timing information is maintained symbolically. However the set of constraints quickly becomes too unwieldy for large circuits. Earlier work in timed state space exploration broadly falls in one of two categories: (a) those that use dense time, and (b) those that consider time as discrete. While the analysis of these two timed models differ, their results coincide in several practical applications if time is discretized at a suitable granularity. Techniques that use dense time mostly employ timed automata [3] based reasoning, and suffer from the associated scalability problems. Techniques, like the one presented in this paper, that use discrete time, enjoy the advantage of being able to encode time directly in a finite number of circuit states, thereby reducing the problem of timed reachability analysis to one of untimed reachability analysis. Efficiently managing predicates on the large number of state variables of the resulting state transition system is a key challenge in this approach.

There are two primary contributions of this paper. The first is a BDD-based algorithm for timed reachability analysis that is inspired by factor graph based algorithms for summary computation. This algorithm exploits locality of interaction of gates and gracefully degrades in accuracy as memory constraints become limiting. This permits us to perform reasonably accurate timed reachability analysis on circuits with more than 1000 gates, when existing algorithms for searching such large state spaces run out of memory. Our second con-



tribution comprises of theoretical arguments and experimental evidence in support of our hypothesis that reasonably accurate timed reachability analysis of digital circuits can be performed with limited main memory.

II. TIMED DIGITAL CIRCUITS

A *timed digital circuit* is a 4-tuple (W, I, O, \mathcal{G}) , where W is a set of wires, $I \subset W$ is the set of primary inputs, $O \subset W$ gives the primary outputs and \mathcal{G} is a set of boolean gates. Every wire has zero delay, carries a boolean value at all times and changes its boolean value instantaneously. A gate $G \in \mathcal{G}$ is a 6-tuple (I_G, O_G, d, l, u, f) , where $I_G \subset W$ is the set of inputs to G , $O_G \in W$ is the output of the gate, $d, l, u \in \mathbb{N}$ are delays associated with the gate, and $f : \{0, 1\}^{|I_G|} \rightarrow \{0, 1\}$ gives the boolean value of the output as a function of boolean values of inputs. We impose the following additional constraints on the circuit structure:

- (a) $\forall G_1, G_2 \in \mathcal{G} (G_1 \neq G_2) \Rightarrow (O_{G_1} \neq O_{G_2})$,
- (b) $\forall w \in W (w \notin I) \Leftrightarrow (\exists G \in \mathcal{G} (w = O_G))$,
- (c) $\forall w \in W (w \notin O) \Leftrightarrow (\exists G \in \mathcal{G} (w \in I_G))$.

Thus, no two gates drive the same wire, and every wire is an input to or output of at least one gate.

A. Discrete-timed model of gates:

Similar to the model used in [2], we model a gate $G = (I_G, O_G, d, l, u, f)$ as a composition of three elements as shown in Fig. II. The *boolean logic block* gives the boolean value of the gate output as a function f of boolean values of its inputs without introducing any delay. The output of this block feeds an *inertial delay element* with a delay of d that models filtering of short input pulses in physical gates. This element changes its output only if a change in its input persists for at least d time units. This output is fed to a *bi-bounded pure delay element* that delays each transition on its input by a non-deterministic delay between l and u . This models RC delays in physical implementations of gates and wires, with delay uncertainties arising from variations in operating conditions, etc. The

behaviour of an inertial delay element with delay d can be modeled by a timed automaton [?] as shown in Fig. II(a). Similarly, a bi-bounded pure delay element with lower and upper bounds l and u can be modeled by a timed automaton as shown in Fig. II(b). In these figures, Zero and One refer to stable *locations*, where the boolean values of *in* and *out* are the same. The primed variables refer to assignments made to the corresponding unprimed variables when transitioning from one location to another.

Throughout this paper, we will assume that *time is discrete*. From the theory of (dense) timed automata [?], we know that for each clock c in a timed automaton, there is a rational constant D_c , such that all values of c greater than D_c are equivalent as far as determining the acceptance of a timed word is concerned. Thus, if time is discretized, the set of possible values of clock c in a timed automaton can be *finitely* represented by discretizing all values from 0 to D_c , and by using a single new value to denote all values of c in excess of D_c . The behaviour of the *discrete-timed* automata of Figs. II(a) and (b) can therefore be represented by finite state automata, with the (finitely many) values of all clocks encoded explicitly in the states.

Given the above model, the state of a gate is defined by values of: (i) boolean variables representing values of the gate inputs, inertial delay element output and gate output, (ii) location variables representing the locations (shown as coloured circles in Fig. II(a) and (b)) of the two discrete-timed automata, and (iii) clock variables of finite enumerated type representing discrete clock values of these automata. We will call these the *state variables of a gate*. A gate changes state by changing values of one or more of these variables consistently with the behaviour of the boolean logic block and the discrete-timed automata. We distinguish between two kinds of state transitions. A *discrete transition* does not advance the clock of any timed automaton, while a *timed transition* increments the clock of every timed automata. Note that a transition that increments a clock of one timed automaton must also increment every clock of every timed automaton that is not reset, since time flows at the same rate for all timed automata.

Although we will use the above model for purposes of our discussion, we note that alternative gate delay models have also been used by researchers to study the timed behaviour of digital circuits []. The techniques presented in this paper can be used for analyzing circuits using such alternative discrete-delay models as well, as long

as the model lends itself to a finite state description with timing information encoded in the states.

B. State transition system of a circuit:

The set of state variables of a timed circuit (W, I, O, \mathcal{G}) is the union of the sets of state variables of all gates in \mathcal{G} . A (timed) state of the circuit is an assignment of values from the appropriate domains to each state variable. A circuit changes state in one of three ways: (a) by having every gate execute a timed transition simultaneously (since time flows at the same rate for all timed automata), (b) by having a gate execute a discrete transition that doesn't change the boolean value of any wire connecting two gates, or (c) by having all gates connected to a wire w simultaneously execute a discrete transition that changes the boolean value of w . A transition of type (c) models the instantaneous propagation of a change in the boolean value of a wire to all gates connected to it. Since transitions of types (b) and (c) do not cause discrete time to elapse, whenever multiple such transitions are enabled in a state, we treat them as concurrently enabled and non-deterministically choose one and execute it. Note that this does not prohibit simultaneous activity in different wires and gates, which can still happen due to transitions of type (a).

Let $T_{a,i}(V_i, V'_i)$ denote the timed transition relation (transitions of type (a)) of gate G_i , where V_i is the set of state variables of G_i and V'_i denotes the next state versions of these variables. Transitions of type (a) for the complete circuit are then given by $T_a(V, V') = \bigwedge_{G_i \in \mathcal{G}} T_{a,i}(V_i, V'_i)$, where $V = \bigcup_{G_i \in \mathcal{G}} V_i$. Let $T_{b,i}(V_i, V'_i)$ be the transition relation that encodes all discrete transitions of type (b) for gate G_i . Transitions of type (b) for the complete circuit can then be represented by $T_b(V, V') = \bigvee_{G_i \in \mathcal{G}} T_{b,i}(V_i, V'_i)$, where the disjunction models the non-deterministic choice of an enabled transition of type (b). Finally, let $Gates(w)$ denote the set of gates connected to wire w , and let $T_{w,i}(V_i, V'_i)$ be the transition relation that encodes all discrete transitions of gate $G_i \in Gates(w)$ that change the boolean value of w . If $V_w = \bigcup_{G_i \in Gates(w)} V_i$, then the discrete transition relation of type (c) modeling the change in value of wire w is given by $T_w(V_w, V'_w) = \bigwedge_{G_i \in Gates(w)} T_{w,i}(V_i, V'_i)$. Transitions of type (c) for the complete circuit are represented by $T_c(V, V') = \bigvee_{w \in W} T_w(V_w, V'_w)$. Since the circuit changes state by executing a transition of type (a), (b) or (c), the overall state transition relation of the circuit is given by $T(V, V') = T_a(V, V') \vee T_b(V, V') \vee T_c(V, V')$.

III. REACHABILITY ANALYSIS

Given a timed digital circuit, we now focus on computing the set of all states reachable from a given initial set of states. Let V be the set of all state variables, and let $S(V)$ be the characteristic predicate of a set of states. The *image* of $S(V)$ under $T(V, V')$ is given by $img(S(V), T(V, V')) = \exists V (S(V) \wedge T(V, V'))$. The set of all reachable states is obtained by starting with the initial set of states, and by repeatedly computing and accumulating the image of the current reachable set under $T(V, V')$ until no new states are obtained. Recalling the definition of $T(V, V')$ from the previous subsection, $img(S(V), T(V, V'))$ is obtained by disjuncting the following: (i) $img(S(V), \bigwedge_{G_i \in \mathcal{G}} T_{a,i}(V_i, V'_i))$, (ii) $img(S(V), T_{b,i}(V_i, V'_i))$ for all gates $G_i \in \mathcal{G}$, and (iii) $img(S(V), T_w(V_w, V'_w))$ for all wires $w \in W$.

Computing each of the above images involves operations on the predicate $S(V)$ that has all state variables in its support set. Timed analysis of even medium-sized circuits (approximately 1000 gates) can cause $|V|$ to reach few tens of thousands, as reported in Section ???. Thus, using a monolithic BDD to represent $S(V)$ leads to serious BDD size blowup problems. While techniques like partitioned OBDDs [], multi-threaded reachability [], searching using high density BDDs [] or hybrid BDD-SAT techniques [] can be used to alleviate the size explosion problem to some extent, we believe there is a complementary approach to addressing this problem. This is based on the observation that each gate or wire in a circuit typically affects a small subset of other gates or wires. Thus, individual transition relations $T_{a,i}(V_i, V'_i)$, $T_{b,i}(V_i, V'_i)$ or $T_w(V_w, V'_w)$ read and update only a small subset of the total set of state variables. This suggests using projections of states on appropriately chosen subsets of V , and optimizing the image computation step using locality of interactions between gates and wires.

Reachability analysis using projections of states has been studied in detail by Cho et al [?] and Govindaraju [4]. Formally, the projection of a set of states $S(V)$ on variables $\Pi \subseteq V$ is computed as $\exists (V \setminus \Pi) S(V)$. Let Π_1, \dots, Π_r be (possibly overlapping) subsets of state variables on which we choose to project the states, and let $\bar{\Pi}_i = V \setminus \Pi_i$ for any subset Π_i of V . In projections-based reachability analysis, we start with projections $S_1(\Pi_1), \dots, S_r(\Pi_r)$ of a set of states and compute the projection

on each Π_j of the image of $\bigwedge_{i=1}^r S_i(\Pi_i)$ under $T_x(V_x, V'_x)$, where x is either a , b_i or w , as discussed above. This projection is computed as $\exists \overline{\Pi_j} \exists V_i (\bigwedge_{k=1}^r S_k(\Pi_k) \wedge T_x(V_x, V'_x))$. Unfortunately, computing $\bigwedge_{k=1}^r S_k(\Pi_k)$ or $T_a(V, V')$ involves predicates on all variables in V , and is computationally expensive in BDD-based tools if $|V|$ is large. Govindaraju et al [4] proposed using a *multiple constrain* operator to partially overcome this problem when computing the image of a conjunction of projections. However, our experiments on timed digital circuits with six thousand and above state variables show that the multiple constrain approach does not scale well, and runs out of memory very soon. The computational bottleneck is largely because multiple constrain eventually computes a predicate on all state variables (tens of thousands), although some intermediate BDD blowups are avoided. This motivates us to investigate new approximate techniques that exploit locality of interaction between gates and wires to compute the projection of an image efficiently.

A. Factor graphs and approximate image computation

As seen above, the basic image computation step in projections based timed reachability analysis is

$$\exists R \bigwedge_{i=1}^n P_i(X_i), \quad (1)$$

where P_i is either a transition predicate or a characteristic predicate of a projection, X_i is the support set (or set of variables whose values affect the value) of P_i , and R is the set of variables being quantified out. For notational clarity, we will henceforth use $\sigma(P_i)$ to denote the support set of P_i , and will omit the support set of a predicate when it is clear from the context. Expression (1) can be viewed as computing the “summary” (over variables in \overline{R}) of a “factored product”. A rich theory for efficiently computing such summaries through *factor graph* representations has been studied in coding theory, machine learning and Bayesian belief propagation. The reader is referred to [?] for an excellent tutorial on this topic. Since boolean formulas with \wedge and \vee form a semiring, all results from [?] apply when computing expression 1. We summarize below key results from this theory that are relevant for our work.

Given a set of variables $V = \{v_1, \dots, v_k\}$ and a set of predicates $\mathcal{P} = \{P_1, \dots, P_n\}$ with $\sigma(P_i) \subseteq V$ for all i , a factor graph \mathcal{F} is an undirected bipartite graph, intuitively representing the “is in support set of” relation. This graph has a *variable node* for each $v_i \in V$ and a *function node* for

each $P_j \in \mathcal{P}$. An undirected edge is drawn from each $P_i \in \mathcal{P}$ to each $v_j \in \sigma(P_i)$. As an example, Fig. ??a shows a factor graph for three predicates. For notational convenience, we will henceforth refer to variable nodes by the (sets of) variables they represent, and function nodes by the predicates represented by them. It is shown in [?] that if the factor graph is cycle-free, an efficient algorithm for computing $\exists V \setminus v_r \bigwedge_{i=1}^n P_i$ can be derived directly from the graph. This is done by considering the factor graph as a tree with root v_i . “Messages” in the form of predicates are sent from the leaves up towards the root in this tree. Every leaf variable node sends the True predicate to its parent, and every leaf function node send its own predicate to its parent. A non-leaf variable node v_j receives predicates from all its children, conjoins them and sends the result to its parent. A non-leaf function node P_l with parent node v_s conjoins all predicates received from its children with itself, projects the conjunction on v_s , and sends the result to its parent. The predicate that finally reaches the root can be shown to be equivalent to $\exists V \setminus v_r \bigwedge_{i=1}^n P_i$. An example of this computation is depicted in Fig. ??b.

The above technique can also be generalized to an efficient *sum-product algorithm* [?] that simultaneously computes projections of $\bigwedge_{i=1}^n P_i$ on every $v_r \in V$. In this algorithm, two “messages” or predicates are sent along every edge, one in each direction. A message sent from a variable node v_i to a neighbouring function node P_j is a conjunction of predicates received by v_i from all neighbouring P_l ’s other than P_j . A message sent from a function node P_i to a neighbouring variable node v_j is the projection on v_j of the conjunction of P_i with all predicates received from neighbouring v_l ’s other than v_j . Since a factor graph is bipartite, every edge e is associated with exactly one variable, say v_e . An important property of the sum-product algorithm is that every message along an edge e is a predicate with the singleton support set $\{v_e\}$. Thus, computing $\exists V \setminus v_r \bigwedge_{i=1}^n P_i$ for all $v_r \in V$ never requires operations on predicates with support set larger than the largest support set of any P_i . If $|\sigma(P_i)|$ is small for each $P_i \in \mathcal{P}$, this gives a memory efficient algorithm for computing projections of $\bigwedge_{i=1}^n P_i$ on all $v_r \in V$, regardless of the size of \mathcal{P} or V .

Factor graphs for projections-based timed reachability analysis of circuits unfortunately abound in cycles. For such graphs, Kschitschang et al [?] suggest several alternative techniques for *exact* summary computation. One alternative, used successfully in coding and machine learning applications,

is to apply the sum-product algorithm regardless of the presence of cycles. This is reported to give good results if the factor graph has a local tree structure. Unfortunately, factor graphs in our application do not have this property. For such cases, Kschitschang et al. suggest two structural transformations, *clustering* and *stretching variable nodes* followed by redundant edge removal, for eliminating cycles. The reader is referred to [?] for details of these transformations. Once all cycles in a factor graph have been eliminated, the sum-product algorithm for cycle-free factor graphs can be applied to the transformed graph to compute exact summaries or projections.

Each of the above transformations results in a factor graph with variable nodes representing *sets of variables*, instead of single variables, in general. Intuitively, this comes close to our application, since in computing expression (1), we must project $\bigwedge_{i=1}^n P_i$ on $V \setminus R$, which could be a set of variables. However, if a variable node represents a large subset of variables, by the property of “messages” passing along an edge mentioned above, all “messages” along any edge connected to this node are predicates with a potentially large support set. Since BDD sizes correlate well with the size of the support set of functions represented by BDDs, it is not desirable to have variable nodes representing large support sets. Preliminary investigations indicate that the stretching transformation results in more variable nodes representing large subsets of variables than that obtained by clustering. While this requires further exploration, in this paper, we choose to focus on clustering for removing cycles in our factor graphs. Clustering either replaces a set of variable nodes by a new variable node, or a set of function nodes by a new function node in the factor graph. All edges in the original factor graph incident on any node in the clustered group are made incident on the new node representing the cluster in the transformed graph. When a set of variable nodes is clustered, the new node represents the set of all variables represented by the clustered nodes in the original graph. When a set of function nodes is clustered, the new node represents the conjunction of all predicates represented by the clustered nodes. Fig. ?? shows an example of clustering that eliminates all cycles in the original factor graph.

There are several ways to cluster nodes for rendering a factor graph cycle-free. In our work, we cluster based on an intuitive and qualitative notion of “relative influence” of variables or predicates in determining the final result. While every variable in

V and every predicate in \mathcal{P} can indeed “influence” the computation of $\exists R \bigwedge_{i=1}^n P_i$, we hypothesize that by and large, variables in $V_0 = V \setminus R$ have the strongest influence on the final result. This is because these variables form the support set of the result predicate. Thus, we wish to cluster variable nodes corresponding to V_0 to represent the intuitively “most influential” cluster of variables. Similarly, all predicates whose support sets have a non-empty intersection with V_0 intuitively have a strong influence on the result. This is because these predicates can directly influence which assignments of truth values to variables in V_0 can cause $\bigwedge_{i=1}^n P_i$ to evaluate to False. For example, suppose v_1 is in $V_0 = V \setminus R$ and P_1 has v_1 in its support set. If P_1 evaluates to False whenever $v_1 = \text{True}$, the projection $\exists R \bigwedge_{i=1}^n P_i$ must also evaluate to False whenever $v_1 = \text{True}$. Predicates whose support sets don’t intersect with V_0 cannot directly influence the result predicate in this manner. Notice that these “influential” predicates correspond to function nodes at a shortest distance of 1 from variable nodes representing V_0 in the factor graph. We choose to cluster these function nodes together, and replace them with a new function node representing the conjunction of all the clustered predicates. Let the predicate representing the conjunction be called L_1 . Fig. ??b shows the result of applying these two clustering transformations to the factor graph of Fig. ??a, where $V_0 = \{v_1, v_2\}$. It can be shown that the above sequence of two clusterings ensures that the transformed graph has *only one edge* incident on the node representing V_0 . This edge connects V_0 to L_1 . It immediately follows that all cycles in the original factor graph containing variable nodes corresponding to V_0 are eliminated. Fig. ??b shows an illustration of this.

The above transformation can be viewed as a first step in converting the original factor graph (with cycles) into a tree with V_0 as the root. Recall that such a tree would immediately give us an algorithm for computing $\exists R \bigwedge_{i=1}^n P_i$, where $V_0 = V \setminus R$. Since a single edge connects V_0 to L_1 in the transformed factor graph, the sum-product algorithm suggests that if the new function node L_1 receives the right “messages” or predicates from all its neighbours other than V_0 , then we could simply conjoin these predicates with L_1 and project the result on $V_0 = V \setminus R$ to obtain $\exists R \bigwedge_{i=1}^n P_i$, the “message” to pass to the root V_0 . Since variable nodes corresponding to V_0 have already been clustered, and since each variable had exactly one corresponding node in the original factor graph, unless V_0 is merged with other variable nodes in the

transformed factor graph, all neighbouring nodes of L_1 other than V_0 do not have any variable in V_0 . Since we are eventually interested in obtaining a tree with V_0 as the root, we disallow V_0 from the merged with other variables nodes

This corresponds to evaluating $\exists R \bigwedge_{i=1}^n P_i$ as $\exists R_0 (L_1 \wedge \exists \bigwedge)$

zz

IV. PROBABILISTIC ANALYSIS OF IMAGE COMPUTATION

We have seen that memory constraints may force Algorithm ?? to overapproximate the effects of some layers when computing expression 1. We now give a probabilistic argument to support our hypothesis that the gain in accuracy by not approximating any layer numbered 0 through k and overapproximating all other layers, decreases on an average as k increases. This justifies being increasingly aggressive in overapproximating the effects of layers with increasing layer numbers as memory constraints become limiting. Our metric for measuring accuracy of computing expression 1 will be the *satisfying fraction* of the resulting predicate, defined as the fraction of all possible assignments to variables in its support set, for which the predicate evaluates to True. Higher the satisfying fraction, the more overapproximate is the result of computing expression 1. Henceforth, we will denote the satisfying fraction of a predicate P as $sf(P)$. It is easy to see that if we overapproximate all layers numbered $k+1$ and higher by the invariantly True predicate, the result of computing expression 1 must logically entail the result obtained by overapproximating all layers numbered k and higher by True. In other words, the satisfying fraction of the former result is no more than that of the latter. What is interesting to study, however, is how the satisfying fraction changes on an average as we increase the largest layer number upto which no overapproximations are made.

In order to do a probabilistic analysis, we need a probability distribution of predicates in each layer. Such a distribution can be obtained, although we don't have enough data at present. In any case, it is non-trivial to characterize such a distribution since it depends on the functionality, delay parameters and interconnection structure of gates in different circuits. To simplify our analysis, we assume that all predicates in each layer are chosen from a *perfectly probable set* as defined below. We believe that this is a weak assumption and results based on this assumption reflect the expected behaviour

when averaged over a large and diverse set of timed circuits.

Definition 4.1: Let $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ be a set of predicates with $\sigma(P_i) \subseteq V$. Then, \mathcal{P} is a *perfectly probable set* (PPS) with parameter f ($0 \leq f \leq 1$), if for every assignment of truth values to variables in V , $f \cdot n$ predicates in \mathcal{P} evaluate to True.

Lemma 4.1: Let P be a predicate drawn uniformly at random from a PPS with parameter f . Then every truth assignment to variables in V makes P True with probability f , and $E[sf(P)] = f$.

Proof: Given in Appendix.

Henceforth we will only consider predicates drawn uniformly at random from perfectly probable sets. Since expression (1) involves projecting a conjunction of predicates, we wish to estimate the expected satisfying fraction of such a projection when the individual predicates are drawn uniformly at random from a PPS, as was assumed earlier to be the case for each layer.

Lemma 4.2: Let $\{P_{i,1}, \dots, P_{i,t}\}$ be t (possibly repeated) predicates chosen uniformly at random from a PPS with parameter f . Let $L = \bigwedge_{j=1}^t P_{i,j}$ and let $\sigma(L) = \{v_1, v_2, \dots, v_s\}$ without loss of generality. Let $L|_r = \exists v_{r+1} \dots \exists v_s L$ be the projection of L on $\{v_1, \dots, v_r\}$. Then, $E[sf(L|_r)] \sim \left(1 - \frac{1}{1+2^{s-r} \cdot f^t}\right)$.

Proof: Given in Appendix.

In order to study how the satisfying fraction of the result of computing expression 1 changes on an average as we increase the largest layer number upto which no overapproximations are made, we use the notation of the previous section and let L_i denote the conjunction of all predicates in layer i , and V_i denote $\sigma(L_{i+1}) \cap \sigma(L_i)$. Let $f_i = sf(L_i)$ (this is the equivalent of f^t in Theorem A.3), $n_i = |\sigma(L_i)|$ and $k_i = |V_{i-1}|$. Let $\mathcal{H}(i)$ denote the expected value of the satisfying fraction of the result of computing expression 1 by overapproximating all layers numbered $i+1$ and higher by True.

Lemma 4.3: $\mathcal{H}(i) = \frac{f_0 \prod_{j=1}^i g_j}{1 + \sum_{j=1}^i \prod_{r=j}^i g_r}$ where $g_j = 2^{n_j - k_j} \cdot f_j$ for each layer L_j .

Proof: Given in Appendix.

To capture the relative gain in accuracy obtained by incrementing the largest layer number that is not overapproximated by True, we define the ratio $T(i) = \frac{\mathcal{H}(i) - \mathcal{H}(i+1)}{\mathcal{H}(i-1) - \mathcal{H}(i)}$. If $T(i) < 1$ for all layers L_i , the relative gain in accuracy forms a decreasing sequence on an average. This condition precisely captures our intuition that overapproximating

| ckt | Gates | Extra gates | Proj. | Variables | N-values | | | Accuracy | | |
|-------|-------|-------------|-------|-----------|----------|------------|---------|----------|-------|-------|
| | | | | | Max | Min | Avg | Max | Min | Avg |
| 74181 | 202 | 130 | 390 | 6242 | 0.940 | 6.866e-119 | 0.00684 | 2.833 | 1.000 | 1.175 |
| 432 | 564 | 368 | 1092 | 17574 | 0.141 | 6.635e-192 | 0.00080 | 2.000 | 1.000 | 1.024 |
| 499 | 724 | 481 | 1407 | 22514 | 0.166 | 4.035e-240 | 0.00075 | 2.000 | 1.000 | 1.023 |
| 880 | 1182 | 739 | 2304 | 36866 | 0.936 | 1.785e-309 | 0.00437 | 2.000 | 1.000 | 1.065 |
| 1355 | 2030 | 768 | 2757 | 44114 | 0.166 | 2.971e-430 | 0.00043 | 2.000 | 1.000 | 1.010 |

TABLE I
ACCURACY COMPARISON

higher numbered layers leads to lower loss of accuracy than overapproximating lower numbered layers. Below, we formulate a sufficient condition for $T(i)$ to be less than 1 for a layers L_i .

Theorem 4.1: Let $N(i) = \sum_{j=1}^i \Pi_{r=j}^i g_r$ for each layer L_i . If $N(i) < 1$. then $T(i) < 1$.

Proof: Given in Appendix.

Thus, if $N(i) < 1$ for all layers i , the relative gain in accuracy decreases on an average for all layers i . In the next section, we give experimental evidence in support of $N(i)$ being less than 1 for all layers i in all the circuits that we analyzed.

V. EXPERIMENTAL RESULTS

In order to evaluate our approach, we have implemented the strategies described in this paper in the public domain reachability analysis engine NuSMV[1]. Our experimental suite consists of gate-level circuits with discrete-time delays. A few small circuits are taken from [2], while several larger circuits were taken from the ISCAS-85 suite. We have converted each circuit to a functionally equivalent one in which each gate has a maximum fanin and maximum fanout of 3. This helps us to reduce the number of local interactions. For each gate, we have used the following delay parameters in the gate delay model of Fig. II: $d = 3, l = 1, u = 2$. To model timed sequences of primary input changes, we non-deterministically decide to toggle the boolean value of every input after every 4 units of time. The initial state for each circuit sets all primary inputs to the boolean value 0, all clocks of all timed automata to 0, and all timed automata start in their “Zero” location (see Fig. II).

The specification for each circuit is given in table IV. Here, *Gates* gives the original number of gates and *Extra gates* gives the number of gates added to convert it to the bounded fanin fanout form discussed above. *Proj* gives the number of projections, and *Variables* represents the total number of state variables.

In Section IV, we had argued that on an average, the incremental contribution of layers to the final projection of an image decreases with increasing

layer number if $N(i) < 1$ for all layers i . We now check to see if $N(i)$ is indeed less than 1 for all circuits used in our experiments. We first note that since projections of images are accumulated during reachability analysis, the satisfying fraction of any layer during an intermediate computation step is overapproximated by the satisfying fraction of the layer after reachability analysis has terminated. This, in turn, is overapproximated by computing the satisfying fraction of a subset of predicates (such that the total number of variable in the support set is less than a threshold) in that layer. Since $N(i) = \sum_{j=1}^i \Pi_{r=j}^i g_r$ and $g_r = 2^{n_r - k_r} f_r$ for all r , using upper bounds of f_r gives upper bounds of $N(i)$ for all layers i . Table IV gives these upper bounds obtained after completing approximate reachability analysis of the larger circuits in our benchmark suite. We observe that the values of $N(i)$ are well below 1 for all layers. This vindicates our conjecture that the lowest numbered layers contribute the most to the projection of an image.

Finally, we also report the accuracy comparison of our approaches. Computing complete set of reachable states requires us to compute the conjunction of reachable states of all projections. This conjunction involves all state variables and hence is prohibitively expensive. We, therefore report the maximum, minimum and average among the ratios of reachable states of respective projections (referred to as columns *Max*, *Min* and *Avg* under *Accuracy* in table IV).

The ratios in the table are not extremely large. On close observation, we have observed that the sets of reachable states in all projections themselves are very small and hence the marginal approximations caused by all our techniques are small.

Performance comparison: We now present and compare experimental results obtained by application of our naive (N) algorithm for $k = 1$ as well as approxImage (AI) algorithms. All our experiments were performed on a 3 GHz Intel(R) Xeon(TM) processor with 4 GB of main memory, and running

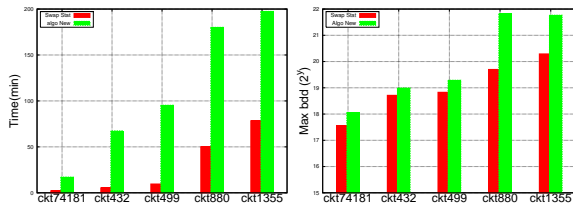


TABLE II

TIME AND BDD NODE PLOTS FOR REACHABILITY ANALYSIS

Fedora Core Linux 2.6.18-1.2798.fc6xen.

The plots in table II show the peak live BDD node count (on a \log_2 scale) and time taken for analyzing each circuit using N and AI . We report results for only the larger circuits. For each of these circuits, Govindaraju and Dill’s method aborted after producing an out of memory exception in NuSMV. The plots show that for every circuit both time and BDD node counts for N are significantly smaller than the corresponding values for AI . The additional time required by algorithm AI is attributable to two factors: (i) conjoining and projecting predicates beyond the 1st layer, and (ii) Since every image computation operation involves only a small number of predicates, we only keep the relevant BDDs in main memory and rest all on the disk. Swapping predicates to and from the disk causes an increase in the total time and takes upto 65% of the total time in certain cases. The increased BDD node counts when using AI are also attributable to two factors: (i) BDDs representing projections of layer $i + 1$ on the support set of layer i need to be maintained when computing the projection of layer i on the support set of layer $i - 1$, and (ii) although the total variable count is always maintained below a threshold, many more BDDs are constructed and manipulated when applying algorithm AI than when applying N .

VI. DISCUSSION AND CONCLUSION

Reachability analysis of discrete-timed digital circuits poses serious scalability challenges because of the large number of state variables required to model delay effects. While exact analysis of circuits containing thousands of gates is beyond the scope of existing BDD-based tools, we have shown in this paper that it is possible to build algorithms that gracefully degrade accuracy-wise when space constraints become limiting. By setting different thresholds in algorithm *approxImgRecur*, as well as by implementing different decomposition strategies for a layer when the threshold is exceeded, we can obtain a family of approximate reachability algorithms for the same problem. We

conjecture that on an average, the loss of accuracy of our algorithm is not much, since the expected incremental contribution of distant layers to increasing the accuracy of computation of a projection of an image is low. Our probabilistic analysis and the analogy to computation of marginals using factor graphs corroborates our hypothesis in this respect. Swapping BDDs in and out of disk allows our technique to scale to arbitrarily large circuits with low fanin and fanout per gate, in principle. While this implies significant time overheads for disk operations, we believe this is a reasonable compromise when analyzing timed circuits with tens of thousands or even millions of state variables. We believe that our approach presents a practical solution to the very important problem of timed analysis of circuits, while allowing the user the ability to increase the accuracy of analysis by relaxing main memory constraints.

REFERENCES

- [1] Cimatti A., Clarke E. M., Giunchiglia F., and Roveri M. NUSMV: A new symbolic model checker. *International Journal on STTT*, 2(4):410–425, 2000.
- [2] Thomas D., Chakraborty S., and Pandya P. Efficient guided symbolic reachability using reachability expressions. *TACAS*, 2005.
- [3] Govindaraju S. G. and Dill D. L. Approximate symbolic model checking using overlapping projections. In *First International Workshop on Symbolic Model Checking (SMC99)*, July 1999. Trento, Italy.
- [4] Govindaraju S. G., Dill D. L., Hu A. J., and Horowitz M. A. Approximate reachability with bdds using overlapping projections. In *DAC*, June 1998.
- [5] Coudert O. and Madre J. C. Implicit and incremental computation of primes and essential primes of boolean functions. In *DAC '92*, pages 36–39. IEEE Computer Society Press, 1992.
- [6] Juvekar S, Taly A, Kanade V, and Chakraborty S. Efficient approximate reachability of network of transition systems. In *Proceedings of the GM RD Workshop on Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems*, Springer, 2007.

APPENDIX

PROOFS OF THE THEOREMS

Lemma A.1: Let P be a predicate drawn uniformly at random from a PPS with parameter f . Then every truth assignment to variables in V makes P True with probability f , and $E[sf(P)] = f$.

a) Proof: Let the total number of possible variable assignments be $\alpha_1 \dots \alpha_n$. Consider any assignment α_i . By definition there would be exactly $f \cdot |PPS|$ predicates in the PPS which would return true on assignment α_i . So probability that a randomly chosen predicate returns True on $\alpha_i = \frac{f \cdot |PPS|}{|PPS|} = f$. Since this is independent of the

assignment, every variable assignment makes the predicate true with probability f .

We define a set of random variables $X_1 \dots X_n$ which can take two values 0 and 1. $X_i = 1$ whenever the variable assignment α_i makes the predicate P true and $X_i = 0$ otherwise. So we have $Pr(X_i = 1) = f$ and $Pr(X_i = 0) = 1 - f$. Since X_i are essentially *Bernoulli* random variables, we have

$$\forall i, E(X_i) = f$$

Let $X = (\sum_1^n X_i)/n$ be a new random variable. It is easy to see that X gives the satisfying fraction of a randomly chosen predicate from the set \mathcal{P} . Therefore by using linearity of expectation we have,

$$\begin{aligned} E(X) &= E((\sum_1^n X_i)/n) = (\sum_1^n (E(X_i)))/n \\ &= (\sum_1^n f)/n = f \end{aligned}$$

Thus, $E[sf(P)] = f$

Lemma A.2: Let \mathcal{P} be a perfectly probable set with parameter f . Consider the set $\mathcal{P}^{(r)}$ of all predicates obtained by choosing r (possibly repeated) predicates from \mathcal{P} and conjoining them. $\mathcal{P}^{(r)}$ is a perfectly probable set with parameter f^r .

b) proof: Consider a predicate P from the set $\mathcal{P}^{(r)}$ and a variable assignment α for it. Let $P = P_1 \wedge \dots \wedge P_r$ where each $P_i \in \mathcal{P}$. P will return True on α iff each P_i returns True on α . Number of predicates in \mathcal{P} which return True on α is exactly $f \cdot |\mathcal{P}|$. So number of ways of choosing r predicates (with repetition) which return True on α is exactly $(f \cdot |\mathcal{P}|)^r$. Therefore number of predicates in $\mathcal{P}^{(r)}$ which return True on α is $(f \cdot |\mathcal{P}|)^r$. So probability that a randomly drawn predicate from $\mathcal{P}^{(r)}$ will return True is $(f \cdot |\mathcal{P}|)^r / |\mathcal{P}^{(r)}| = f^r$. Thus $\mathcal{P}^{(r)}$ forms a perfectly probable set with parameter f^r .

Lemma A.3: Let $\{P_{i,1}, \dots, P_{i,t}\}$ be t (possibly repeated) predicates chosen uniformly at random from a PPS \mathcal{P} with parameter f . Let $L = \bigwedge_{j=1}^t P_{i,j}$ and let $\sigma(L) = \{v_1, v_2, \dots, v_s\}$ without loss of generality. Let $L|_r = \exists v_{r+1} \dots \exists v_s L$ be the projection of L on $\{v_1, \dots, v_r\}$. Then, $E[sf(L|_r)] \sim \left(1 - \frac{1}{1+2^{s-r} \cdot f^t}\right)$.

c) Proof: By lemma A.2 since \mathcal{P} is a PPS, $\mathcal{P}^{(t)}$ is also a PPS with parameter f^t . Thus L is a predicate drawn uniformly at random from the set $\mathcal{P}^{(t)}$. So, by lemma A.1, the probability that L return True on a particular variable assignment α is f^t . Let R be the set of all possible predicates of the form $\exists v_{r+1} \dots \exists v_s L$ where $L \in \mathcal{P}^{(t)}$. Since \mathcal{P} is perfectly probable, R should also be perfectly

probable. Let α_r be the projection of α on the variables $v_1 \dots v_r$. Further let f' be the probability that the assignment α_r returns true for a predicate chosen uniformly at random from \mathcal{R} . So $1 - p'$ is the probability that the assignment α_r returns false. An assignment α_r would make the predicate $L|_r$ false iff the corresponding predicate L returns false for the set of all those assignments which overlap with $\alpha|_k$ for variables $v_1 \dots v_r$. Number of such assignments is $2^{(s-r)}$. So the probability that P would return false for these 2^{s-r} assignments is $\sim (1 - f^t)^{2^{s-r}}$. Hence the probability that the assignment $\alpha|_k$ would return true for the predicate $L|_k \sim 1 - f' \leq (1 - f^t)^{2^{s-r}}$. Therefore

$$f' \sim 1 - (1 - f^t)^{2^{s-r}} \sim 1 - \frac{1}{(1 + f^t)^{2^{s-r}}} \quad (2)$$

Since s is very large we can replace the term $(1 + f^t)^{2^{s-r}}$ by its first order approximation $1 + 2^{s-r} \cdot f^t$. Using Lemma A.1 we get $E[sf(L|_r)] = f' \sim 1 - \frac{1}{1+2^{s-r} \cdot f^t}$ \square

Theorem A.1: If $\mathcal{H}(i)$ is the expected value of the satisfying fraction of *naiveApprox*(R, \mathcal{P}, i) then $\mathcal{H}(i) = \frac{f_0 \prod_{j=1}^i g_j}{1 + \sum_{j=1}^i \prod_{r=j}^i g_r}$ where $g_i = 2^{n_i - k_i} f_i$ for all i .

d) Proof: We prove this result by induction.

Base Case: $i = 0$. Here the result $\mathcal{H}(i) = f_0$ is trivially true.

Induction Hypothesis: Assume the result for $i = t$. So we have $\mathcal{H}(t) = \frac{f_0 \prod_{j=1}^t g_j}{1 + \sum_{j=1}^t \prod_{r=j}^t g_r}$.

Inductive Step: We go from layer t to $t + 1$. Let L_t and L_{t+1} be the predicates for layers t and $t + 1$ respectively. In order to compute the effect of layer $t + 1$, we compute the satisfying fraction of the predicate obtained by taking conjunction of L_t and projection of L_{t+1} on variables in $\sigma(L_t)$. This forms our new predicate for layer t . Let f'_t be the expected satisfying fraction of this predicate. Now we simply replace f_t by f'_t in \mathcal{H}_t and get the value for \mathcal{H}_{t+1} .

Observe that $\exists \sigma(L_{t+1}) / \text{supp} L_t (L_t \wedge L_{t+1}) = L_t \wedge \exists \sigma(L_{t+1}) / \sigma(L_t) L_{t+1}$

Therefore $f'_t = E[sf(L_t)] \times E[sf(L_t \wedge \exists \sigma(L_{t+1}) / \sigma(L_t) L_{t+1})]$. According to our model $\sigma(L_{t+1}) / \sigma(L_t)$ has k_{t+1} variables and L_{t+1} is predicate over n_{t+1} variables with satisfying fraction f_{t+1} . So by theorem A.3 we get satisfying fraction of $\exists \sigma(L_{t+1}) / \sigma(L_t) L_{t+1}$ is

$1 - \frac{1}{1+2^{(n_{t+1}-k_{t+1})} \cdot f_{t+1}}$. Thus we have

$$\begin{aligned} f'_t &= f_t \times \left(1 - \frac{1}{1+2^{(n_{t+1}-k_{t+1})} \cdot f_{t+1}}\right) = \frac{f_t \cdot g_{t+1}}{1+g_{t+1}} \\ g'_t &= \frac{g_t \cdot g_{t+1}}{1+g_{t+1}} \end{aligned}$$

$$\begin{aligned}
\mathcal{H}(t+1) &= \mathcal{H}(t)|_{g_t \rightarrow g'_t} \\
&= \frac{f_0 \left(\prod_{j=1}^{t-1} g_j \right) \cdot \frac{g_t \cdot g_{t+1}}{1+g_{t+1}}}{1 + \sum_{j=1}^t \left(\prod_{r=j}^{t-1} g_r \right) \cdot \frac{g_t \cdot g_{t+1}}{1+g_{t+1}}} \\
&= \frac{f_0 \prod_{j=1}^{t+1} g_j}{1 + g_{t+1} \sum_{j=1}^t \prod_{r=j}^{t-1} g_r g_{t+1}} \\
&= \frac{f_0 \prod_{j=1}^{t+1} g_j}{1 + \sum_{j=1}^{t+1} \prod_{r=j}^{t+1} g_r}
\end{aligned}$$

Thus the result is also true for $t+1$. Induction $\forall t$ completes the proof. \square

Theorem A.2: Let $N(i) = \sum_{j=1}^i \prod_{r=j}^i g_r$ for each layer L_i . If $N(i) < 1$, then $T(i) < 1$.

e) Proof: Denote $\prod_{j=1}^i g_j$ by $K(i)$. We have the following relations $\forall i$

$$\begin{aligned}
N(i+1) &= (1 + N(i)) * g_{i+1} \\
K(i+1) &= K(i) * g_{i+1}
\end{aligned}$$

Therefore, $\forall i, K(i)(1 + N(i+1)) - K(i+1)(1 + N(i)) = K(i)$. So we have

$$\begin{aligned}
\frac{\mathcal{H}(i) - \mathcal{H}(i+1)}{\mathcal{H}(i-1) - \mathcal{H}(i)} &= \frac{\frac{K(i)}{1+N(i)} - \frac{K(i+1)}{1+N(i+1)}}{\frac{K(i-1)}{1+N(i-1)} - \frac{K(i)}{1+N(i)}} \\
&= \frac{\frac{K(i)(1+N(i+1)) - K(i+1)(1+N(i))}{(1+N(i))(1+N(i+1))}}{\frac{K(i-1)(1+N(i)) - K(i)(1+N(i-1))}{(1+N(i-1))(1+N(i))}} \\
&= \frac{\frac{K(i)}{(1+N(i+1))}}{\frac{K(i-1)}{(1+N(i-1))}} \\
&= \frac{K(i)(1 + N(i-1))}{K(i-1)(1 + N(i+1))} \\
&= \frac{K(i-1)g_i(1 + N(i-1))}{K(i-1)(1 + N(i+1))} \\
&= \frac{N(i)}{1 + N(i+1)}
\end{aligned}$$

If $N(i) < 1$ then $\frac{N(i)}{1+N(i+1)} < 1$. Thus the ratio $T(i)$ is strictly less than 1 $\forall i$ \square