ON-LINE MINIMIZATION OF TRANSITION SYSTEMS (Extended Abstract)

David Lee

Mihalis Yannakakis

AT&T Bell Laboratories
Murray Hill, New Jersey

ABSTRACT

We are given a transition system implicitly through a compact representation and wish to perform simultaneously reachability analysis and minimization without constructing first the whole system graph. We present an algorithm for this problem that applies to general systems, provided we have appropriate primitive operations for manipulating blocks of states and we can determine termination; the number of operations needed to construct the minimal reachable graph is quadratic in the size of this graph. We specialize the method to obtain efficient algorithms for extended finite state machines that apply separable affine transformations on the variables.

1. INTRODUCTION

There has been a lot of work towards developing automatic methods for analyzing communication protocols and validating that they are free of errors, such as deadlock, unspecified reception, or unexercised code. The usual approach involves a state exploration to discover all reachable system states [Ho1, W]. The automatic verification of more involved properties, such as temporal logic specifications by model checking, can be viewed also as an augmented reachability analysis in a graph that combines the program and the specification [CVWY, CES, VW]. Despite the advances, the well known *state explosion* problem imposes a severe limit on the size of the protocols that can be analyzed automatically. The problem is that a protocol (generally any program) is expressed in a compact notation. When we go to an explicit representation of the state space, for example by enumerating all possible values for the variables, then a dramatic increase in size results (and this may not be possible if some domains are infinite). Although the reachable part of the state space is usually a small portion of the total space, there are still quite severe limitations. An idea that has been emerging in the last few years is to represent the state space symbolically instead of explicitly, that is, to use an implicit representation for classes of states and to explore many of them at once [Br, BCMDH, CBM].

Although the full state graph is huge, many of its states may be equivalent to each other, thus analyzing them separately is redundant work. There are well known efficient algorithms for minimizing deterministic and nondeterministic systems [Hp, PT], however, they also assume an explicit representation of the state space. In this paper we address the problem of performing reachability and minimization at the same time. Our goal is to do work that is bounded by a function of the number N of reachable equivalence classes. Although simple reachability problems for any sort of nontrivial compact notation become easily PSPACE-hard for finite domains and undecidable for infinite domains, and the number N in general can be exponential or unbounded, we still have to develop algorithms for validating large protocols. Thus, we have to identify the parameters and methods that will permit us to push further the limits of how "large" is large, and we believe that complexity analysis has a role to play in this regard to focus the direction. In practice, N is expected to be significantly smaller than either the number of reachable states by itself or the total number of (reachable and unreachable) equivalence classes.

Another application of the problem that we study is in protocol conformance testing. The difference between validation and testing is that the former is supposed to check the correctness of the design, whereas the latter has to check that the implementation conforms to the design (see [Ho2] for a more thorough discussion). Testing even an explicitly represented finite state machine (FSM) is not a trivial problem [SL, YL]. Formal testing methods typically break a protocol into its "control" portion, and its "data" portion that includes variables, parameters etc, and design a test for the first part that is an ordinary FSM. One could extend in principle the testing to the data portion by expanding the values of the variables, but then we face again the same state explosion problem. Controlling this problem to any extent can help in this respect. Indeed, Holzmann has implemented an early version of our algorithm at Bell Labs for the testing of SDL specified protocols for the 5ESS switching system, improving significantly the quality of the test generated, compared to the previous control-based methods [private communication].

We summarize now the results and the organization of the rest of this abstract. Section 2 covers the basics of transition systems and equivalence relations, discusses briefly some examples, and provides a minimization algorithm. In Section 3 we present an algorithm for computing the reachable part of the minimized system. We assume that blocks of states are represented symbolically, and we list the primitive operations that are needed. The algorithm arrives at this reachable minimal graph after $O(N^2)$ operations. However, in the absence of any other information, the algorithm may not know it has found the final graph, and may continue working. In general this is unavoidable unless we have a routine that solves the static problem of verifying that a given graph is the desired one; clearly, if we cannot solve the static problem of verifying the answer, we cannot solve the dynamic problem of computing it either. In Sections 4 and 5 we apply the method to extended finite state machines with "separable" affine transitions; i.e., machines with variables that are modified via separate affine transformations. In Section 4 we examine the case of arbitrary real coefficients and show that the termination problem of verifying the result graph can be reduced to a Linear Programming problem with two variables per inequality, which admits a strongly polynomial solution. In Section 5 we study the case of integral coefficients, and show that the termination problem can also be solved in polynomial time.

2. TRANSITION SYSTEMS AND THEIR MINIMIZATION

In this section we present a minimization algorithm after describing transition systems and equivalence.

Definition 2.1. A *transition system* is a tuple $\Theta = (Q, \pi, I, T)$ consisting of (1) a set of *configurations* Q; (2) a partition π of Q; (3) a finite set I of actions (or inputs); and (4) a set T of transition relations on Q corresponding to the actions, i.e., for each action $a \in I$, there is a relation $R_a \subseteq Q \times Q$. The transition system is deterministic if the transition relation for every action is a function, otherwise it is nondeterministic. □

For simplicity, we assume that the actions are fully specified, i.e., for each input a and configuration $p \in Q$ there exists $q \in Q$ such that $(p, q) \in R_a$. For clarity, we did not include outputs in the definition of a transition system; the theory can be easily generalized to the case with outputs.

The set of configurations in the above definition may be infinite. We think of a transition system as a labelled graph whose nodes are the configurations and which has an arc from node (configuration) p to node q labelled by action a if $(p,q) \in R_a$. If S is a set of configurations, we use the notation $a(S)$ for $\{ q \mid \exists p \in S.\ (p,q) \in R_a \}$, and $a^{-1}(S)$ for $\{ p \mid \exists q \in S.\ (p,q) \in R_a \}$. We define also another graph for a transition system, the *quotient graph*, as follows. The nodes are the blocks of the partition π of the system. There is an arc from block B to block C labelled by action a if and only if $(p,q) \in R_a$ for some $p \in B$ and $q \in C$. This is the graph representing the quotient transition system Θ/π whose configurations are the blocks of π.

A finite action sequence $\alpha \in I^*$ represents the composition R_α of the transition relations in the sequence (for α the empty string, R_α is the identity). We say that a configuration p has an α-path to configuration q (or the sequence α *maps* p to q) if $(p,q) \in R_\alpha$.

Definition 2.2. Given a transition system $\Theta = (Q, \pi, I, T)$, let $\equiv$ be the largest equivalence relation among the configurations, such that $p \equiv q$ implies:
1. p and q belong to the same block of the partition π;

2. For every input $a \in I$, (i) if a maps p to p' then a maps q to some configuration q' with $q' \equiv p'$; and conversely, (ii) if a maps q to q' then a maps p to some configuration p' with $p' \equiv q'$.
Two configurations $p, q \in Q$ are *equivalent* if $p \equiv q$.
The *minimized* (or *reduced*) transition system of Θ is the system $\Theta' = (Q, \pi', I, T)$, where π' is the partition induced by the equivalence relation among the configurations. □

This is the usual notion of *bisimulation equivalence*. In the case of deterministic systems, Definition 2.2 is equivalent to the following definition: the configurations p, q are equivalent if, for any action sequence $\alpha \in I^*$, the sequence α maps p to a configuration of block B of the partition π, if and only if it maps q to a (possibly different) configuration of B.
We will usually identify the minimized system with its quotient Θ'/π'.

Examples: Transition systems are usually represented implicitly.
1. Figure 1 depicts a system consisting of n processes sharing a resource, and an allocator coordinating exclusive access to it [Pn]. The processes and the allocator communicate through Boolean variables g_i, r_i. An arc $s \rightarrow t$ in the figure with label a/b (a, b literals) indicates that the process i or the allocator can move from state s to state t provided the literal a is true, and the transition has the effect of setting b to true. Transitions with - in the first part, namely $1 \rightarrow 2$ and $3 \rightarrow 4$ in process i, do not have any attached precondition. The critical section where a process uses the resource is state 3.

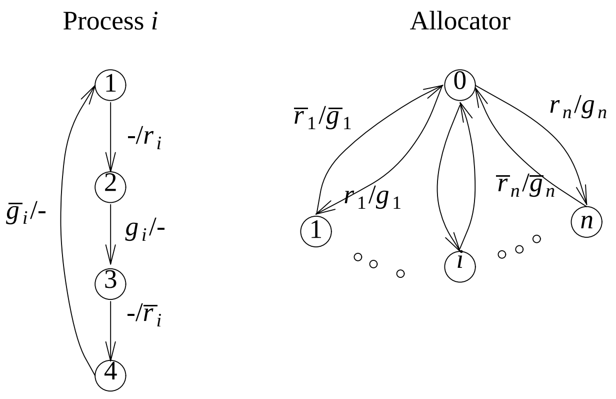

Figure 1

A configuration in this system consists of a choice of state for each process and for the allocator, and an assignment of a Boolean value to each variable, i.e., there are $(n+1)16^n$ configurations. The set of actions is trivial (i.e., a singleton). There is a transition from configuration p to q if one process or the allocator makes a move, the corresponding move is enabled according the values of the variables in p, and the values in q stay the same except for the indicated effect of the move. The choice of the partition π is determined by the property we want to validate about the system, i.e., which configurations we care to distinguish. For example, let π consist of three blocks containing respectively those configurations where 0, 1, or at least 2 processes are in state 3 (i.e., using the resource). Since π is symmetric with respect to renaming the processes, it is not hard to see that the minimized system has a polynomial number of blocks. Intuitively, any two configurations that can be obtained from each other by renaming the processes are equivalent. Thus, it does not matter exactly which state each process is in, and precisely what the variable values are; all that matters are the total counts. A similar fact holds if we want to verify a property (i.e., start with a partition π) that concerns one process (or a constant number); such as, "can Process 1 starve?" Clearly, the same remark applies to all symmetric systems of this type where identical processes interact with each other, either directly or through a central controller.

2. It is possible that a transition system is infinite but the minimized system is finite. Real time systems are modeled in [ACD] by *timed graphs*. These are essentially finite state machines extended with a finite number of timers taking real values; transitions may have enabling preconditions (comparisons of timer values to integer constants) and may reset some of the timers; see [ACD] for the detailed definition. A configuration in this case consists of a state and an assignment of real values to the timers, thus there is a continuum

of configurations. Nevertheless, Alur et al showed that one can do model checking in this model by reducing it to a problem on a finite graph. Let π be the partition where two configurations p, q are in the same block if and only if they have the same state component and have enabled exactly the same transitions of the timed graph. This partition induces a notion of "transition-equivalence" among the configurations, which turns out to have a finite number of classes. Alur et al [ACD] do model checking by constructing a finite transition system which is (a refinement of) the minimized system.

3. Similarly, communication protocols can be modeled by *extended finite state machines* (EFSM), which are machines augmented with a finite number of variables (eg. Boolean, arithmetic); a transition may have an input, an enabling precondition on the variables, and have as effect the change of state, production of output and a transformation on the values of the variables. An EFSM is a compact representation of a transition system obtained by expanding all possible values of the variables. One can define a partition π as above and study properties instead in the minimized transition system. □

Let Θ be a transition system, and consider its quotient graph. We say that an arc, $B \rightarrow C$ with label a, of the quotient graph is *stable* if every configuration of B has an a-arc to some configuration of C; otherwise, it is unstable. The transition system is stable if all arcs of its quotient graph are stable. An important property is that, every unstable transition system has a unique coarsest stable refinement, and that refinement is precisely the reduced transition system

We can obtain easily the reduced transition system by splitting until there are no unstable arcs. In the algorithm below, we have a list L of blocks that have possibly unstable arcs into them. We use the fact that if there are no unstable arcs into block B, then this property will continue to hold if we refine the partition as long as B is not split.

Algorithm 2.1. (Transition System Reduction)
Input: Unstable transition system $\Theta = (Q, \pi, I, T)$;
Output: The reduced transition system $\Theta' = (Q, \pi', I, T)$.

```
begin
    L := list of the blocks of π;
    while (L ≠ ∅) do
        remove a block B from L;
        for every block C in current partition and action a do
            $C' := C \cap a^{-1}(B)$;
            if ($C' \neq \emptyset$ and $C' \neq C$) then do
                remove block C from the partition and from the list L (if present);
                add blocks C′ and $C'' = C - C'$ to the partition and to L;
            end
        end
    end
end
```
□

To carry out the minimization we need the following *Basic Operations* on blocks in π: (i) Represent the intersection of two blocks; (ii) Represent the inverse of a block $a^{-1}(B)$ (actually we only need the combination $C \cap a^{-1}(B)$ of (i) and (ii)); (iii) Represent the difference of two blocks: $C - C'$ and (iv) Test for emptiness. Assume for now that each operation takes time c.

Theorem 2.1. If Algorithm 2.1 does not terminate, then the reduced transition system has infinitely many blocks. If it does terminate then it constructs the reduced transition system (Q, π', I, T) in time $O(ckN^2)$, where $k = |I|$ is the number of actions and $N = |\pi'|$ is the number of blocks in the reduced system. □

In the case of a finite transition system that is explicitly given, the straightforward implementation of the above algorithm has complexity $O(mn)$, where n is the number of configurations and m is the total number of arcs [KS]. In this case it is well known that one can do better: Hopcroft used the "process the smaller half" idea to develop a minimization algorithm [Hp] for deterministic machines with complexity

$O(kn\log n)$ and Paige and Tarjan obtained a bound of $O(m\log n)$ for the nondeterministic case [PT]. Note that the number n in these complexity bounds is really the total number of configurations while N in our bound $O(ckN^2)$ is the number of equivalent classes of configurations, which could be significantly smaller than n. It is not obvious that an improvement over our time bound is possible when we have implicit representations. We conjecture that it is possible; this is an interesting data structure problem. However, it is outside our main focus here, which is how to perform reachability and minimization at the same time.

3. REACHABILITY OF TRANSITION SYSTEMS

We have a transition system $\Theta = (Q, \pi, I, T)$ and an initial configuration p_0. Let π' be the partition corresponding to the equivalence relation of the system, let Θ/π' be the quotient graph of the minimized system, and let G_0 be the subgraph of Θ/π' that is reachable from the block containing the initial configuration p_0; we call this the *reachable minimal graph*. We want to compute G_0. By definition, for each configuration q that is reachable from p_0 in the transition system Θ, the graph G_0 contains the block of π' containing q. Conversely, if a block B of π' is in G_0 then it contains some configuration q that is reachable from p_0; however, note that it may contain also other configurations that are not reachable.

If w is a (finite or infinite path) in Θ or in G_0, let its *projection* on π, denoted $\Pi(w)$ be the sequence whose ith element is the block of π containing the ith node of the path. Since π' is a refinement of π, the projection operation is well defined. If α is a sequence of actions, let $\Pi_\Theta(\alpha)$ be the set of projections $\Pi(w)$ of all α-paths w in Θ that start at p_0, and define $\Pi_0(\alpha)$ to be the set of projections of the paths in G_0 starting at the block containing p_0. The graph G_0 has the following property.

Lemma 3.1. Let α be a sequence of actions. Then $\Pi_\Theta(\alpha) = \Pi_0(\alpha)$. □

For a sequence of actions, no matter it is in Θ or G_0, their projections on π are identical. Thus, if the initial partition π reflects adequately the features we need to distinguish in the configurations, for the sake of the properties of the computations we want to verify, then it suffices to consider the graph G_0 instead of the full transition system Θ. Note that for this, we only need the graph; we do not need to compute exactly the blocks.

The two obvious ways for constructing the reachable minimal graph are: (1) compute first all the configurations that are reachable from p_0, and then minimize the transition system induced on them; (2) minimize first the given transition, and then compute the part that is reachable from the block of the initial configuration. Both of these methods can be arbitrarily bad. In general, the reachable minimal graph can be arbitrarily smaller than both the reduced system and the number of reachable configurations, which are the minimal amount of work to be done using the two obvious methods, respectively. Furthermore, the reachable minimal graph can be finite while the other two can be infinite.

We have to use an intermediate method that explores the graph and splits blocks simultaneously, combining the forward inference of reachability information with the backward inference of inequivalence information. The only previous algorithm for the problem is the one proposed by Bouajjani et al [BFH, BFHRR]; however, it does not meet our goals. When we want to compute the whole minimal system, it does not matter too much how we split blocks according to unstable transitions. Here it can make a big difference. In Figure 2 we show two simple examples of an extended finite state machine with one variable x; the domain of the variable is $[0,M]$, with M large. There are no preconditions on the transitions, except that the image is in the domain. Here a configuration consists of a state and a value for the variable x. Each transition transforms x as shown on the label. In the initial configuration the system is in state s_0 with $x = 0$. In both cases the reachable minimal graph is the same as the one of the figure but the interval of x in each block is smaller. Suppose we start from the block of the initial configuration, and we work our way top-down, splitting blocks as we go along to stabilize the initial portion of the system we have seen (this is the strategy of [BFH, BFHRR]). Suppose that if both arcs out of a block are unstable, we always stabilize first the left one labelled x (these arcs could be paths instead). Then it is easy to see that the time $T(i)$ to stabilize the first i levels according to this strategy obeys the recurrence $T(i+1) = 2T(i)+i+1$, thus if there are $n+1$ levels, then $T(n) \geq 2^n$. What happens is that we keep cutting off parts of the intervals associated with the states; all these parts are in different blocks of the reduced system, but they are not reachable, so all this work is wasted. What we should have done instead is go to the bottom of the graph, and then work our way up fixing all the intervals one by one in linear time. This example is acyclic. In general of course in the presence of cycles there is no top and bottom.

In the example of Figure 2b with similar conventions, if x ranges over the real numbers in $[0, M]$ and we try to stabilize the root block, we will cycle forever. However, if we split once the block of s_1 we would discover that there are only two reachable configurations: the initial $(s_0, x=0)$ and $(s_1, x=0)$.

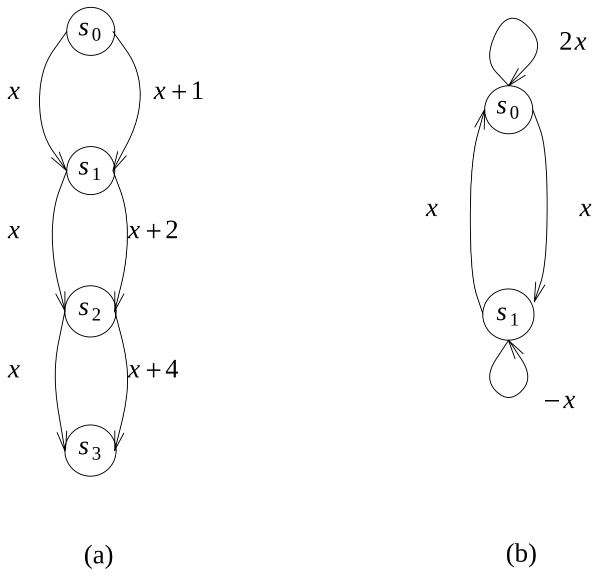

Figure 2

There are two simple basic ideas in our approach: we keep track of some configurations reached from the initial one, and we prefer to search forward than split, but we do not search unless we know for sure we are accessing inequivalent configurations; second, we do not split blocks unless we know they are reachable, and we give every reachable block a fair chance to split. During the execution of the algorithm, we have reached some blocks which we mark, and pick one reachable configuration p_B from each such block B.

Definition 3.1. A transition system $\Theta = (Q, \pi, I, T)$ is *marked* if one or more blocks have a *marked configuration*. A block B with the marked configuration p_B is denoted by $<B, p_B>$. The *marked graph* of Θ has a node for each marked block, and has an edge from block B to block C labelled by an action a if $a(p_B) \cap C \neq \emptyset$. □

Note that the marked graph of Θ is generally not an induced subgraph of the quotient graph.

Definition 3.2. Let $\Theta = (Q, \pi, I, T)$ be a marked transition system, and $\Theta' = (Q, \pi', I, T)$ the reduced system. Let H be the quotient graph of Θ' and let H' be its subgraph induced by those blocks that contain marked configurations or are reachable from them. We say that Θ is *semi-stable*, if its marked graph is equal to H'. □

Semi-stability means that the marked graph G of Θ is the "right" graph as far as the part of the system that is reachable from marked configurations is concerned, except some of the blocks are not completely refined yet.

Lemma 3.2. Let $\Theta = (Q, \pi, I, T)$ be a marked transition system. The system is semi-stable if and only if (1) no marked configuration has an arc to a nonmarked block, and (2) for any path from $<B, p_B>$ to $<B', p_{B'}>$ in the marked graph the corresponding transition sequence f has the property that $f(p_B) \in B'$. □

In order to be able to compute the minimal reachable graph, at the very least we must be able to recognize when we are done, when we have obtained the right graph, i.e., the system is semi-stable. We show that the dynamic problem of building the graph reduces to this static problem.

Algorithm 3.1. (Reachable Minimal Graph)

Input: A transition system (Q, π, I, T) with an initially marked block $<B_0, p_0>$.
Output: The minimal reachable graph and a semi-stable transition system (R, ρ, I, T).

```
begin
    STACK := $\emptyset$; /* blocks to search from */
    QUEUE := $\emptyset$; /* unstable blocks to be split in a FIFO order */
    mark $<B_0, p_0>$;
    push($<B_0, p_0>$, STACK);
search:
    while (STACK $\neq \emptyset$) do /* depth-first search and mark reachable blocks */
        $<B, p> := pop(STACK)$;
        for (each action a in I) do
            for (each $\tilde{B}$ such that $\tilde{B} \cap a(p) \neq \emptyset$ ) do /* a block containing configurations of $a(p)$ */
                if ($B \cap a^{-1}(\tilde{B}) \neq B$ and B is not in QUEUE) then do /* B unstable */
                    insert ($<B, p>$, QUEUE);
                end
                if ($\tilde{B}$ is not marked) then do /* $\tilde{B}$ has never been searched */
                    select $p_{\tilde{B}} \in \tilde{B} \cap a(p)$;
                    mark $<\tilde{B}, p_{\tilde{B}}>$;
                    add edge $[a : <B, p> \rightarrow <\tilde{B}, p_{\tilde{B}}>]$;
                    push($<\tilde{B}, p_{\tilde{B}}>$, STACK); /* for further search */
                end
                else do /* $\tilde{B}$ has been searched and marked $<\tilde{B}, p_{\tilde{B}}>$ */
                    add edge $[a : <B, p> \rightarrow <\tilde{B}, p_{\tilde{B}}>]$;
                end
            end
        end
    end
split:
    while (QUEUE $\neq \emptyset$) do /* split unstable blocks */
        $<B, p> := delete(QUEUE)$;
        $B' := B$;
        for (each action a in I) do /* stabilize B by splitting */
            for (each $\tilde{B}$ such that $\tilde{B} \cap a(p) \neq \emptyset$ ) do /* a block containing configurations of $a(p)$ */
                $B' := B' \cap a^{-1}(\tilde{B})$; /* split B to stabilize a */
            end
        end
        $B'' := B - B'$;
        $B := B'$;
        add block $B''$ to the partition;
        for (each edge $[a: <C, q> \rightarrow <B, p>]$ in the marked graph) do /* incoming transitions */
            if ($a(q) \cap B \neq \emptyset$) then do /* edge is still valid*/
                if ($C \cap a^{-1}(B) \neq C$ and $<C, q>$ is not in QUEUE) then do /* C unstable */
                    insert ($<C, q>$, QUEUE);
                end
            end
            else do /* $a(q) \cap B = \emptyset$ and edge becomes invalid */
                delete edge $[a: <C, q> \rightarrow <B, p>]$;
            end
            if ($a(q) \cap B'' \neq \emptyset$ ) then do
```

```
                if (B'' is not marked) then do
                    select $p_{B''} \in a(q) \cap B''$ ;
                    mark $<B'', p_{B''}>$;
                    push($<B'', p_{B''}>$, STACK);
                end
                add edge $[a: <C, q> \rightarrow <B'', p_{B''}>]$;
                if ($C \cap a^{-1}(B'') \neq C$ and $<C, q>$ is not in QUEUE) then do /* C unstable */
                    insert ($<C, q>$, QUEUE);
                end
            end
        end
        if ($STACK \neq \emptyset$) then do
            goto search;
        end
        if (system is semi-stable) then do
            goto terminate;
        end
    end
terminate:
    return the marked graph and marked blocks $\rho$;
end □
```

During the whole processing, configurations in different blocks are not equivalent. Let N be the number of blocks in the reachable reduced system (R, ρ, I, T). At any moment of execution of Algorithm 3.1, there are three classes of blocks: (i) marked block, which contains one or more reachable blocks in ρ; (ii) unmarked block, which contains one or more reachable blocks in ρ; (iii) unmarked block, which is disjoint from R. The sum of the Class (i) and (ii) blocks is no more than N.

Lemma 3.3. Consider a point in the execution of Algorithm 3.1. Assume that the number of current Class (i) blocks is less than N and that there are K blocks in *QUEUE*. Then after executing the split-**while** loop no more than K times, we have at least one more Class (i) or (ii) block. □

Let again c be the time required to perform each of the basic operations. Suppose we do not use the semistability test.

Theorem 3.1. Given a transition system (Q, π, I, T) with an initially marked block $<B_0, p_0>$, suppose that the reachable reduced system (R, ρ, I, T) is finite and let N be its number of blocks. Then Algorithm 3.1 constructs the minimal reachable graph and a semi-stable transition system within time $O(ckN^2)$ where k is the number of actions. □

If the reduced transition system (including the unreachable part) is finite, then it is easy to see that Algorithm 3.1 will terminate in finite time. In general however, if only the reachable part is finite, then the algorithm may not know that it has constructed the right graph and terminate the execution unless we know N, the number of blocks in the reachable reduced system, or we can determine whether the current system is semi-stable. Suppose that we have a subroutine *TERMINATE*, which determines whether the current system is semi-stable. Then

Corollary 3.1. Assume that subroutine *TERMINATE* determines if the current system is semi-stable in time $q(N)$. Then we can compute the minimal reachable graph and terminate in time $O(ckN^2 + q(N))$.

Sketch of Proof. Let $t(n)$ be the time to construct a marked graph with n nodes. We run *TERMINATE* only when $t(n_i) \geq \sum_{j=1}^{i} q(n_j)$ where n_i is the number of nodes of the system at the ith time we call *TERMINATE*. That is, we run it only when the construction part has paid for its cost, so it gets amortized. Clearly, we cannot hope to do better than $q(N)$. □

Note that the final semi-stable system may not be identical to the reachable reduced system; each block of the semi-stable system contains exactly one block of the reachable reduced system but may be a proper superset. However, from Definition 3.2 and Lemma 3.1, the graph suffices. In the full paper we will discuss several variations of the basic algorithm that achieve the same goal. We omit them here for lack of space, since they do not improve the worst-case complexity.

If we do not have available a semistability test, then the algorithm may run forever. However, we can show a kind of competitiveness result indicating that we are not doing too badly given the primitives at hand. More specifically, consider any algorithm which works also by splitting; i.e. starting from the initial partition, it picks at each iteration (by some arbitrary strategy) a block B and splits it according to the one step transition, until finally at some point it chooses some blocks from the current partition and declares them to be the reachable reduced system. Let us call such an algorithm *generic*.

Corollary 3.2. Suppose that there is a generic algorithm that terminates in a finite amount of time, say after m splits it delivers a reachable reduced system with N blocks. Then Algorithm 3.1, without the semistability test, terminates with the reachable reduced system also in finite (at most $O(cmN)$) time. □

Note that the algorithm of [BFHRR] does not satisfy Theorem 3.1 and the corollaries, as the simple examples in Figure 2 show.

4. REAL AFFINE TRANSITION SYSTEMS

In the following two sections, we specialize our approach to systems with affine transitions. We can determine the semi-stability of the current system to terminate the algorithm; in some cases, we can even compute the reachable reduced system efficiently. Obviously, the transitions are deterministic.

An *affine transformation* f from R^r to R^r is of the form $y_i = f_i(\vec{x}) = \sum_{j=1}^{r} \alpha_{i,j} x_j + \beta_i$, $i = 1, \cdots, r$, where $\vec{x} = [x_1, \cdots, x_r]^T \in R^r$, $\vec{y} = [y_1, \cdots, y_r]^T \in R^r$, and $\alpha_{i,j}, \beta_i \in R$. The transformation is *separable* if there is at most one variable on the right-hand side of each equation; it is *strongly separable* if the ith variable maps to the ith variable, i.e., $y_i = \alpha_{i,i} x_i + \beta_i$, otherwise it is weakly separable. For general affine transformations, we do not know how to compute the minimal reachable system. We will show how to do it efficiently for separable transformations. The separability assumptions are not too restrictive. For practical systems such as IEEE standard 802.2 Logical Link Control [LLC91] Protocol, the arithmetic operations typically only involve separable variables; in fact even separability is too general, for example they have small integral coefficients, a case we examine in the next section. In this section we consider the case of real coefficients.

4.1. One Variable

We consider an affine transition system (Q, π, I, T) represented by an extended finite state machine as in Figure 2. Thus, a configuration consists of a state and a set of real values for the variable x. Each block of the partition π is assumed to consist of a given state component and an interval $[a, b]$ for the variable x (for example, this is the case if the partition originated from the preconditions on the transitions of an EFSM which compare variable values to constants). Each transition in T applies an affine transformation on x, which may depend on the block of the partition.

Consider the application of Algorithm 3.1. For all the Basic Operations, the intersection of intervals and the inverse image of an interval remain an interval. The difference of two intervals may not be an interval. For instance, for $a<b<c<d$, $[a, d]-[b, c] = [a, b] \cup [c, d]$. When we stabilize an interval by splitting, the new block may become a union of two intervals. We claim that we can make the two intervals two separate blocks for further processing since points (configurations) in the two intervals are not equivalent. Take two points $u \in [a, b]$ and $v \in [c, d]$. Suppose on the contrary that they are equivalent and belong to the same interval in the original partition. Then for any affine transition sequence they must be mapped into the same interval in the original partition. Since affine transitions are monotonic and the original blocks are intervals, $[u, v]$ must be mapped into the same interval and all the points in $[u, v]$ must be equivalent to u and v. On the other hand, $[b, c] \subset [u, v]$ but are not equivalent to u and v. This contradiction completes the proof of the claim.

Assume that the current system has an initially marked interval $< [a_0, b_0], p_0 >$. After completing search and mark, we want to determine if the system is semi-stable. For each marked interval block

$[a_i, b_i]$ with marked configuration (point) p_i, we introduce two auxiliary variables: u_i and v_i, $i = 1, \cdots, N$, where N is the number of nodes in the marked graph. We impose constraints: $a_i \le u_i \le p_i \le v_i \le b_i$ for $i = 0, \ldots, N$. For each transition with transformation $x := \alpha_{j,i} x + \beta_j$ from $[a_i, b_i]$ to $[a_j, b_j]$, we have constraints: $\alpha_{j,i} u_i + \beta_j \ge u_j$ and $\alpha_{j,i} v_i + \beta_j \le v_j$ if $\alpha_{j,i} \ge 0$, or $\alpha_{j,i} u_i + \beta_j \le v_j$ and $\alpha_{j,i} v_i + \beta_j \ge u_j$ if $\alpha_{j,i} < 0$.

Lemma 4.1. The system is semi-stable if and only if the above LP problem has a feasible solution. If it does have a feasible solution, then the block intervals $[u_i^*, v_i^*]$ of the reachable reduced system can be computed by optimizing the objective function $\sum_i (v_i - u_i)$. □

For the purpose of computing the minimal reachable graph, we only need the feasibility decision. If the LP is not feasible and the routine returns a contradictory subset of constraints, we can show that one can use the information to construct efficiently a path along which we can expand the search, i.e., reach new blocks and refine the partition instead of just continuing to split blindly; we defer the details to the full paper, since it does not improve the worst case complexity.

Each Basic Operation takes constant time now. The LP problem has only two variables involved in a constraint, and there are $2N$ variables and $(2k + 3)N + 1$ constraints, where k is the number of actions. It can be solved in time $O(kN^2(log\ k + N \log^2 N))$ [CM]. From Corollary 3.1, we see that the LP cost dominates in this case.

4.2. Several Separable Variables

Consider $r \ge 2$ variables. Variables are *separable* if each initial block in π is a *cylinder* $B = \prod_{i=1}^{r} [a_i, b_i]$ and each transition involves only two variables. In the case of strongly separable variables, coordinates are totally separated, and the problem is essentially one-dimensional. We formulate one LP problem to determine if the current system is semi-stable.

In the case of weakly separable transitions, there may be some interaction between the variables. While we search and construct the marked graph, we also compute a *dependency* graph as follows. For each node B_i in the marked graph, we have a cluster of r nodes: $(B_{i,1}, \cdots, B_{i,r})$, each of which corresponds to a variable. For an edge in the marked graph, we add r edges in the dependency graph. More specifically, for each transition $y_p = \alpha_{p,q} x_q + \beta_p$, we add an edge from $B_{i,p}$ to $B_{j,q}$, $1 \le p, q \le r$. We formulate one LP problem for the dependency graph as in the one variable case. We summarize:

Theorem 4.1.. For an affine real transition system with r separable variables and initial cylinder blocks, we can construct the minimal reachable graph in time $O(rkN^2(log\ k + N \log^2 N))$, where k is the number of actions and N is the number of nodes of the graph. □

5. INTEGRAL AFFINE TRANSITIONS

An affine transition is integral if all the coefficients and variable values are integers. We consider again separable variables, and assume that the blocks of the initial partition are intervals or cylinders. In the integral case we can decide semi-stability in linear time. Thus, from Corollary 3.1 we have:

Theorem 5.1. For an integral affine transition system with r separable variables and initial cylinder blocks, the minimal reachable graph can be constructed in time $O(rkN^2)$, where k is the number of actions and N is the number of nodes in the graph. □

We sketch below the basic ideas for the linear time semi-stability algorithm for the one variable case. Given a cycle C (not necessarily simple), we can compose the transformations along the edges of the cycle yielding a combined transformation $\alpha x + \beta$, also affine with integral coefficients; we say C is an α cycle. We say that the cycle is *expandable* if it does not survive in the minimal reachable graph (thus, the system is not semi-stable). We summarize in the next lemma some conditions.

Lemma 5.1. 1. If C is an α cycle of the marked graph and $|\alpha| > 1$, then the cycle is expandable unless every marked configuration of a block of the cycle maps to the marked configuration of the next block, and then in the final refinement every block will be reduced to a singleton configuration.
2. If C is an $\alpha = 1$ cycle and $\beta \ne 0$ then it is expandable.
3. Assume that there are two different $\alpha = -1$ cycles passing through the same node u: $-x + \beta_1$ and

$-x + \beta_2$ with $\beta_1 \neq \beta_2$ (we call these inconsistent cycles); then at least one of them is expandable. □

The next lemma shows how to find expandable cycles.

Lemma 5.2. Consider the subgraph consisting of the $|\alpha| = 1$ transitions, and let S be a strongly connected component. Given an arbitrary node u of S, there are expandable $\alpha = 1$ cycles or inconsistent $\alpha = -1$ cycles in S if and only if there are such cycles passing through u. □

The algorithm works as follows. Ignore for the time being the $\alpha = 0$ edges. Compute the strongly connected components of the subgraph H induced by the rest of the edges. If an SCC has an $|\alpha| > 1$ edge, then it must have a $|\alpha| > 1$ cycle. Reduce the blocks of the SCC and of its ancestors in H to singletons consisting of the marked configuration; check that the mappings are preserved.

Consider an SCC with only $|\alpha| = 1$ edges. Choose an arbitrary node u and grow a depth-first search tree rooted at u. At the same time, we compute two function values at each node as follows. Function SH_+ is the composite shift of an $\alpha = 1$ path from u to the node, and SH_- is the composite shift of an $\alpha = -1$ path from u to the node. Initially $SH_+(u) = 0$; $SH_-(u)$ is undefined. We assign the values in the depth-first order. For a newly searched node w from the tree edge (t, w), we assign SH values according to the transition on (t, w): $\alpha x + \beta$. (i) $\alpha = 1$: $SH_+(w) = SH_+(t) + \beta$; $SH_-(w) = SH_-(t) + \beta$. (ii) $\alpha = -1$: $SH_+(w) = -SH_-(t) + \beta$; $SH_-(w) = -SH_+(t) + \beta$. For a nontree edge transition (t, w), we also compute the $SH(w)$ values from this transition and compare them with the SH values already computed at node w (if defined). If they are the same we continue search and assign SH values. Otherwise, there are discrepancies and we claim that there must be expandable cycles. Such a cycle can be found by a path from w to root u.

Lemma 5.3. In the SCC there are neither $\alpha = 1$ cycles with $\beta \neq 0$ nor inconsistent $\alpha = -1$ cycles passing through u if and only if the SH values are consistent at every node. □

After processing the $|\alpha| \geq 1$ edges, if this subgraph passes the test, then we compute an interval for each node using the SH values. Finally, we add the $\alpha = 0$ edges (these are assignments) and check that they are consistent with the intervals we computed.

We can extend the algorithm to the case of many separable variables using the dependency graph as defined in the previous section.

6. CONCLUSIONS

We have introduced a simple and powerful general technique for minimizing transition systems on the fly, a problem which at the beginning appeared to us as too vague and ill-defined to formulate and prove any rigorous complexity results about it. We reduced the dynamic problem of generating the minimal reachable graph to the static problem of recognizing it when we obtained it, and applied the approach to the case of arithmetic variables with affine separable transformations. In fact, we started this research the other way around; first we looked at cases of affine transformations, and then as we were extending the algorithms, they were becoming more messy, which made clear the need for a more general tool.

There is a number of interesting open problems raised by this work. We list some below.
1. Besides the arithmetic variables we considered, Boolean variables form the other important part of protocol specifications. There has been work on symbolic representation and manipulation of predicates on Boolean variables, for example, by Binary Decision diagrams [Br, BCMDH]. Can we say anything quantitative here? How does our approach extend to Boolean variables?
2. If we want to extend the solution to general affine transformations, there are two problems. One has to do with the fact that the number of constraints needed to represent the blocks will grow. The second problem is that we do not know an algorithm for detecting termination; this is an interesting problem concerning forming a fixed point of a Linear Program.
3. The bottleneck in the integral affine case was in the dynamic part. Improving on this part involves maintaining the strongly connected components of the graph dynamically as it grows and old edges may be redirected. Not much is known about this problem in general. In our case we conjecture that the problem can be solved in $O(N \log N)$ time.

7. REFERENCES

[ACD] Alur, R., Courcoubetis, C., and Dill, D. [1990] Model Checking for Real Time Systems, in *Proc. 5th IEEE LICS*.

[BFH] Bouajjani, A., Fernandez, J-C., and Halbwachs, N. [1990]. Minimal Model Generation, in *Proc. 2nd Workshop on Computer-Aided Verification*, DIMACS Series Vol. 3, ACM-AMS, pp. 85-91.

[BFHRR] Bouajjani, A., Fernandez, J-C., Halbwachs, N., Raymond, P. and Ratel, C. [1991]. Minimal State Graph Generation, *IMAG Grenoble Preprint*.

[Br] Bryant, R. E. [1986]. Graph-based Algorithms for Boolean Function Manipulation, in *IEEE Trans. on Computers*, vol. 35.

[BD] Budkowski, S. and Dembinski, P. [1987]. An Introduction to Estelle: a Specification Language for Distributed Systems, in *Computer Networks and ISDN Systems* **14**, pp. 3-23.

[BCMDH] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. J. [1990] Symbolic Model Checking: 10^{20} States and Beyond, in *Proc. 5th IEEE LICS*, pp. 428-439.

[CES]

[CM] Cohen, E. and Meggido, N. [1991]. Improved Algorithms for Linear Inequalities with Two Variables per Inequality, in *Proc. 23rd Annual ACM Symp. on Theory of Computing*, pp. 145-155.

[CBM] Coudert, O., Berthet, C., and Madre, J. C. [1989]. Verification of Synchronous Sequential Machines Based on Symbolic Execution, in *Proc. Intl. Workshop on Automatic Verification Methods for Finite State Systems*, LNCS 407, Springer Verlag.

[CVWY] Courcoubetis, C., Vardi, M., Wolper, P. and Yannakakis, M. [1990]. Memory Efficient Algorithms for the Verification of Temporal Properties, in *Proc. 2nd Workshop on Computer-Aided Verification*, DIMACS Series Vol. 3, ACM-AMS.

[GS] Graf, S. and Steffen B. [1991]. Compositional Minimization of Finite State Systems, in *Proc. 2nd Workshop on Computer-Aided Verification*, DIMACS Series Vol. 3, ACM-AMS, pp. 57-73.

[Ho1] Holzmann, G. J. [1988]. An Improved Protocol Reachability Analysis, in *Software, Practice and Experience*, vol. 18, pp. 137-161.

[Ho2] Holzmann, G. J. [1991]. *Design and Validation of Protocols,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

[Hp] Hopcroft, J. E. [1971]. An $n \log n$ Algorithm for Minimizing States in a Finite Automaton, in *Theory of Machines and Computations*, Z. Kohavi and A. Paz (eds.), pp. 189-196, Academic Press, New York.

[KS] Kanellakis, P. and Smolka, S. [1983]. CCS Expressions, Finite State Processes and Three Problems of Equivalence, in *Information and Computation*, vol. 86, pp. 43-68.

[LLC] Logical Link Control [1989]. *International Standard ISO 8802-2, IEEE Std. 802.2,* The Institute of Electrical and Electronics Engineers, Inc.

[PT] Paige, R. and Tarjan, R. [1987]. Three Partition Refinement Algorithms, in *SIAM J. on Computing*, vol. 16, pp. 973-989.

[Pn]

[SL] Sidhu, D. P. and Leung, T.-K. [1989]. Formal Methods for Protocol Testing: A Detailed Study, in *IEEE Trans. on Soft. Eng.*, vol 15, pp. 413-426.

[Si] Sifakis, J. [1982]. A Unified Approach for Studying the Properties of Transition Systems, in *Theoretical Computer Science***3.**

[VW] Vardi, M. Y. and Wolper, P. [1986]. An Automata-Theoretic Approach to Automatic Program Verification, in *Proc. LICS, pp. 322-331.*

[W] West, C. H. [1978] Generalized Technique for Communication Protocol Validation, in *IBM J. Res. and Devel.,* vol. 22, pp. 393-404.

[YL] Yannakakis, M. and Lee, D. [1991]. Testing Finite State Machines, in *Proc. of the 23rd Annual ACM Symposium on Theory of Computing*, pp. 476-485.