Optimization of Lyapunov Invariants in Verification of Software Systems

Mardavij Roozbehani, *Member, IEEE*, Alexandre Megretski, *Member, IEEE*, and Eric Feron *Member, IEEE*

***Abstract*—The paper proposes a control-theoretic framework for verification of numerical software systems, and puts forward software verification as an important application of control and systems theory. The idea is to transfer Lyapunov functions and the associated computational techniques from control systems analysis and convex optimization to verification of various software safety and performance specifications. These include but are not limited to absence of overflow, absence of division-by-zero, termination in finite time, absence of dead-code, and certain user-specified assertions. Central to this framework are Lyapunov invariants. These are properly constructed functions of the program variables, and satisfy certain properties—analogous to those of Lyapunov functions—along the execution trace. The search for the invariants can be formulated as a convex optimization problem. If the associated optimization problem is feasible, the result is a certificate for the specification.**

***Index Terms*—Software Verification, Lyapunov Invariants, Convex Optimization.**

I. Introduction

SOFTWARE in safety-critical systems implements complex algorithms and feedback laws that control the interaction of physical devices with their environments. Examples of such systems are abundant in aerospace, automotive, and medical applications. The range of theoretical and practical challenges that arise in analysis, design, and implementation of safety-critical software systems is extensive, see, e.g., [1], [2], [3], and the references therein. While safety-critical software must satisfy various resource allocation, timing, scheduling, and fault tolerance constraints, the foremost requirements are that it must be free of run-time errors, and when expected, terminate in finite time. The main objective of this paper is to present a systematic framework for verification of these properties.

A. Overview of Existing Methods

In this section, we provide a brief overview of formal verification methods, as well as system theoretic methods, noting that a sharp contrast defining the boundaries between these methods does not exist.

Manuscript received July 30, 2011; revised March 30, 2012; accepted August 29, 2012. Date of publication March xx, 2013; date of current version August 27, 2012. This work was supported by the National Science Foundation, Grants CNS-1135955 and CPS-1135843, The Army Research Office under MURI Award W911NF-11-1-0046, NASA under Grant/Cooperative Agreement NNX12AM52A, and by Siemens AG, Munich, Germany. Recommended for publication by Associate Editor Paulo Tabuada.

Mardavij Roozbehani and Alexandre Megretski are with the Laboratory for Information and Decision Systems (LIDS), Department of Electrical Engineering and Computer Science (EECS), Massachusetts Institute of Technology, Cambridge, MA. E-mails: {mardavij, ameg}@mit.edu. Eric Feron is with the School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA. E-mail: feron@gatech.edu.

1) Formal Methods: Formal verification methods are model-based techniques [4], [5], [6] for proving or disproving that a mathematical model of a software (or hardware) satisfies a given *specification*, i.e., a mathematical expression of a desired behavior. The approach adopted in this paper too, falls under the category of model-based verification methods. Herein, we briefly review *model checking*, *abstract interpretation*, and some of the related methods.

a) Model Checking (MC): In model checking [7] the system is modeled as a finite state transition system, e.g., automata [8] or timed automata [9], [10], and the specifications are expressed in some form of logic formulae, e.g., temporal or propositional logic [11]. The verification problem then reduces to a graph search, and symbolic algorithms are used to perform an exhaustive exploration of all possible states. MC has proven to be a powerful technique for verification of circuits [12], security and communication protocols [13], [14] and stochastic processes [15].

For software systems, when applicable, MC techniques result in strong statements about the behavior of the system. The trade-off is in the increased computational requirements and limited scalability to large systems. The advent of Binary Decision Diagrams [16] and SAT solvers [17], which are efficient data structures for representing and solving boolean satisfiability problems, significantly improved the scalability of these techniques. An important trend in MC, supported by recent advances in Satisfiability Modulo Theories (SMT) solvers [18], is formulation of verification problems as SMT problems. These advances have improved the scope and power of MC, as well as alternative verification methods that reduce to MC. Nevertheless, when the program has non-integer variables, or when the state space is continuous, MC is not directly applicable. In such cases, combinations of various abstraction techniques, e.g., relational abstractions and constraint-based approaches, and MC have been proposed [19], [20], [21], [22]; scalability, however, remains a challenge.

b) Abstract Interpretation (AI): Abstract Interpretation is a theory for formal approximation of the *operational semantics* of computer programs in a systematic way [23]. Construction of abstract models involves abstraction of domains—typically in the form of a combination of sign, interval, polyhedral, and congruence abstractions of sets of data—and functions that operate on the domains. A system of fixed-point equations is then generated by symbolic forward/backward executions of the abstract model. An iterative equation solving procedure, e.g., Newton's method is used for solving the nonlinear system of equations, the solution of which results in an inductive invariant assertion, which is then used for checking

the specifications. In practice, to guarantee finite convergence of the iterates, narrowing (refining outer approximations) operators are constructed to estimate the solution, followed by widening (expandding inner approximations) to improve the speed of convergence to the estimate [24]. This compromise can be a source of conservatism in analysis [25]. Nevertheless, these methods have been used in practice for verification of (limited) properties of embedded software of commercial aircraft [26], [27]. With time, several extensions and instantiations of AI have been developed. For example, [28] uses AI for the analysis of numerical errors in floating-point arithmetic, while [29] focuses on index array variables. Over time, AI has also benefitted from significant improvements, notably via improved solutions to the fixed-point equations [30], and the development of many new abstract domains, some of which have obvious links to system-theoretic methods [31].

c) Other methods: Alternative formal methods can be found in the computer science literature mostly under *deductive verification* [11], *type inference* [32], and *data flow analysis* [33]. These methods share extensive similarities with AI in that a notion of program abstraction and symbolic execution or constraint propagation is present in all of them. Further details and discussions of the methodologies can be found in [5] and [24]. More recently, optimization and constraint satisfaction methods [34] have gained increased attention as the primary mechanisms to carry the computations associated with these methods. Program analysis engines combining several of the aforementioned techniques have also been developed and have been successful in verifying device derivers in real-world applications [35].

2) System Theoretic Methods: While software analysis has been the subject of an extensive body of research in computer science, treatment of the topic in the control systems literature is more recent yet growing fast.

The relevant results in the systems and control literature can be found in the field broadly described as hybrid systems. However, the actual range of system models developed and used by the systems community extends far beyond that, with a systematic emphasis on the development of models that combine expressivity with problem tractability. Linear systems [36] naturally embody this tendency. With the advent of many different models with various tractability properties, significant efforts were devoted to drawing bridges linking these different models. One-way bridges include the abstraction mechanisms used in nonlinear and robust system analyses as early as 120 years ago [37], and then revisited by the nonlinear and robust control community [38], [39] and sometimes linked to other well-known concepts in system theory [40]. Two-way bridges linking equivalent models can be found in the extensive literature dealing with bisimulations [41], [42], [43].

Many of the available techniques for safety verification of hybrid systems are explicitly or implicitly based on computation of reachable sets, either exactly or approximately. These include but are not limited to techniques based on quantifier elimination [44], ellipsoidal calculus [45], and mathematical programming [46]. Alternative approaches aim at establishing properties of hybrid systems through barrier certificates [47], numerical computation of Lyapunov functions [48], [49], or by combined use of bisimulation mechanisms and Lyapunov techniques [50], [51], [19].

Inspired by the concept of Lyapunov functions in stability analysis of nonlinear dynamical systems [37], [52], in this paper we propose Lyapunov invariants for analysis of computer programs. While Lyapunov functions and similar concepts have been used in verification of stability or temporal properties of system level descriptions of hybrid systems [53], [48], [49], to the best of our knowledge, this paper is the first to present a systematic framework based on Lyapunov invariance and convex optimization for verification of a broad range of code-level specifications for computer programs [1]. Accordingly, it is in the systematic integration of new ideas and some well-known tools within a unified software analysis framework that we see the main contribution of our work, and not in carrying through the proofs of the underlying theorems and propositions. The introduction and development of such framework provide an opportunity for the field of control to systematically address a problem of great practical significance and interest to both computer science and engineering communities. The framework can be summarized as follows:

1) Dynamical system interpretation and modeling (Section II). We introduce generic dynamical system representations of programs, along with specific modeling languages which include Mixed-Integer Linear Models (MILM), Graph Models, and Mixed-Integer Linear over Graph Hybrid Models (MIL-GHM).
2) Lyapunov invariants as behavior certificates for computer programs (Section III). Analogous to a Lyapunov function, a Lyapunov invariant is a real-valued function of the program variables, and satisfies a *difference inequality* along the trace of the program. It is shown that such functions can be formulated for verification of various software specifications that can be formalized as unreachability or finite-time termination.
3) A computational procedure for finding the Lyapunov invariants (Section IV). The procedure is standard and constitutes these steps: (i) Restricting the search space to a linear subspace. (ii) Using convex relaxation techniques to formulate the search problem as a convex optimization problem, e.g., a Linear Program (LP) [59], or a Semidefinite Program (SDP) [60], including SDP relaxations of semi-algebraic problems [61], [62], [63]. (iii) Using convex optimization software for numerical computation of the certificates.

II. Dynamical System Interpretation and Modeling of Computer Programs

We interpret computer programs as discrete-time dynamical systems and introduce generic models that formalize this interpretation. We then introduce MILMs, Graph Models, and MIL-GHMs as structured cases of the generic models. The specific models are used for computational purposes.

[1]This paper constitutes the synthesis and extension of ideas and computational techniques expressed in the workshop [54], and subsequent conference papers [55], [56] and [57]. Some of the ideas presented in [54], [55], and [56] were independently reported in [58].

A. Generic Models

1) Concrete Representation of Computer Programs: We will consider generic models defined by a finite state space set X with selected subsets $X_0 \subseteq X$ of initial states, and $X_\infty \subset X$ of terminal states, and by a set-valued state transition function $f : X \mapsto 2^X$, such that $f(x) \subseteq X_\infty, \forall x \in X_\infty$. We denote such dynamical systems by $\mathcal{S}(X, f, X_0, X_\infty)$.

Definition 1: A program $\mathcal{P}$ is a set of sequences with elements from a given set (e.g., a subset of $\mathbb{R}^n$). The dynamical system $\mathcal{S}(X, f, X_0, X_\infty)$ is a $\mathcal{C}$-representation of a computer program $\mathcal{P}$, if the set of all sequences in $\mathcal{P}$ is equal to the set of all sequences $\mathcal{X} = (x(0), x(1), \dots, x(t), \dots)$ of elements from X, satisfying

$$x(0) \in X_0 \subseteq X, \qquad x(t+1) \in f(x(t)) \qquad \forall t \in \mathbb{Z}_+ \quad (1)$$

The uncertainty in $x(0)$ allows the program to depend on initial conditions, and the uncertainty in f models dependence on parameters, as well as the ability to respond to real-time inputs.

Example 1: **Integer Division** (adopted from [4]): The functionality of Program 1 is to compute the result of the integer division of dd (dividend) by dr (divisor).

```
int IntegerDivision ( int dd, int dr )
{int q = {0}; int r = {dd};
while (r >= dr)
{   q = q + 1;
    r = r − dr; }
return r; }
```

Program 1: The Integer Division Program

A concrete dynamical system model ($\mathcal{C}$-representation) of Program 1 is constructed by defining the following elements:

$$X = \underline{\mathbb{Z}}^4, \text{ where } \underline{\mathbb{Z}} = \mathbb{Z} \cap [-32768, 32767]$$
$$X_0 = \{(\mathrm{dd}, \mathrm{dr}, \mathrm{q}, \mathrm{r}) \in X \mid \mathrm{q} = 0,\ \mathrm{r} = \mathrm{dd}\}$$
$$X_\infty = \{(\mathrm{dd}, \mathrm{dr}, \mathrm{q}, \mathrm{r}) \in X \mid \mathrm{r} < \mathrm{dr}\}$$
$$f : (\mathrm{dd}, \mathrm{dr}, \mathrm{q}, \mathrm{r}) \mapsto (\mathrm{dd}, \mathrm{dr}, \mathrm{q}, \mathrm{r}) + u$$
$$u = \begin{cases} (0, 0, 1, -\mathrm{dr}), & (\mathrm{dd}, \mathrm{dr}, \mathrm{q}, \mathrm{r}) \in X \backslash X_\infty \\ (0, 0, 0, 0), & (\mathrm{dd}, \mathrm{dr}, \mathrm{q}, \mathrm{r}) \in X_\infty \end{cases}$$

Note that if $\mathrm{dd} \geq 0$, and $\mathrm{dr} \leq 0$, then the program never exits the "while" loop, eventually leading to an overflow. The program terminates if both dd and dr are positive.

2) Abstract Representation of Computer Programs: In a $\mathcal{C}$-representation, the elements of the state space X belong to a finite subset of the set of rational numbers that can be represented by a fixed number of bits in a specific arithmetic framework, e.g., fixed-point or floating-point arithmetic. When the elements of X are non-integers, due to the quantization effects, the set-valued map f often defines very complicated dependencies between the elements of X, even for simple programs involving only elementary arithmetic operations. An abstract model over-approximates the behavior set in the interest of tractability. The drawbacks are conservatism of the analysis and (potentially) undecidability. Nevertheless, abstractions in the form of formal over-approximations make it possible to formulate computationally tractable, sufficient conditions for a verification problem that would otherwise be intractable.

Definition 2: Given a program $\mathcal{P}$ and its $\mathcal{C}$-representation $\mathcal{S}(X, f, X_0, X_\infty)$, we say that $\overline{\mathcal{S}}(\overline{X}, \overline{f}, \overline{X}_0, \overline{X}_\infty)$ is an $\mathcal{A}$-representation, i.e., an *abstraction* of $\mathcal{P}$, if $X \subseteq \overline{X}$, $X_0 \subseteq \overline{X}_0$, and $f(x) \subseteq \overline{f}(x)$ for all $x \in X$, and the following condition holds:

$$\overline{X}_\infty \cap X \subseteq X_\infty. \quad (2)$$

Thus, every trajectory of the actual program is also a trajectory of the abstract model. For proving Finite-Time Termination (FTT), we need to be able to infer that if all trajectories of $\overline{\mathcal{S}}$ eventually enter $\overline{X}_\infty$, then all trajectories of $\mathcal{S}$ will eventually enter X_∞. Requiring that $\overline{X}_\infty \subseteq X_\infty$ may not be possible as X_∞ is often a discrete set, while $\overline{X}_\infty$ is often dense in the domain of real numbers. The definition of $\overline{X}_\infty$ as in (2) resolves this issue.

Construction of $\overline{\mathcal{S}}(\overline{X}, \overline{f}, \overline{X}_0, \overline{X}_\infty)$ from $\mathcal{S}(X, f, X_0, X_\infty)$ involves abstraction of each of the elements $X,\ f,\ X_0,\ X_\infty$ in a way that is consistent with Definition 2. Abstraction of the state space X often involves replacing the domain of *floats* or integers or a combination of these by the domain of real numbers. Abstraction of X_0 or X_∞ often involves a combination of domain abstractions and abstraction of functions that define these sets. For instance, the sign function:

$$\mathrm{sgn}(x) = 1\!\!1_{[0,\infty)}(x) - 1\!\!1_{(-\infty,0)}(x),$$

which may appear explicitly or in modeling *if-then-else* blocks in computer programs [64], can be abstracted as follows:

$$\overline{\mathrm{sgn}}(x) \in \{v \mid xv \geq 0,\ v \in \{-1, 1\}\}.$$

We are often interested in such polynomial set-valued abstractions. As another example, we show that the absolute value function over the domain $[-1, 1]$ can be represented (precisely) in the following way:

$$\mathrm{abs}(x) = \{xv \mid x = 0.5(v + w), (w, v) \in [-1, 1] \times \{-1, 1\}\}$$

Interested readers may refer to [64] for semialgebraic set-valued abstractions of some commonly-used nonlinearities including trigonometric functions, abstractions of modular arithmetic operations, and also abstractions of fixed-point and floating point operations based on ideas from [65].

B. Specific Models of Computer Programs

Specific modeling languages are particularly useful for automating the proof process in a computational framework. Here, three specific modeling languages are proposed: *Mixed-Integer Linear Models (MILM), Graph Models*, and *Mixed-Integer Linear over Graph Hybrid Models (MIL-GHM).*

1) Mixed-Integer Linear Model (MILM): Proposing MILMs for software modeling and analysis is motivated by the observation that by imposing linear equality constraints on boolean and continuous variables over a quasi-hypercube, one can obtain a relatively compact representation of arbitrary piecewise affine functions defined over compact polytopic subsets of Euclidean spaces (Proposition 1). The earliest

reference to the statement of universality of MILMs appears to be [66], in which a constructive proof is given for the one-dimensional case. A constructive proof for the general case is given in [64].

Proposition 1: **Universality of Mixed-Integer Linear Models.** Let $f : X \mapsto \mathbb{R}^n$ be a piecewise affine map with a closed graph, defined on a compact state space $X \subseteq [-1,1]^n$, consisting of a finite union of compact polytopes. That is:

$$f(x) \in 2A_i x + 2B_i \qquad \text{subject to} \qquad x \in X_i,\ i \in \mathbb{Z}(1,N)$$

where, each X_i is a compact polytopic set. Then, f can be specified precisely, by imposing linear equality constraints on a finite number of binary and continuous variables ranging over compact intervals. Specifically, there exist matrices F and H, such that the following two sets are equal:

$$G_1 = \{(x, f(x)) \mid x \in X\}$$
$$G_2 = \Big\{(x,y) \mid F\begin{bmatrix} x & w & v & 1 \end{bmatrix}^T = y, H\begin{bmatrix} x & w & v & 1 \end{bmatrix}^T = 0, (w,v) \in [-1,1]^{n_w} \times \{-1,1\}^{n_v} \Big\}.$$

Mixed Logical Dynamical Systems (MLDS) with similar structure were developed in [67] for analysis of a class of hybrid systems. The main contribution here is in the application of the model to software analysis. A MIL model of a computer program is defined via the following elements:

1) The state space $X \subset [-1,1]^n$.
2) Letting $n_e = n + n_w + n_v + 1$, the state transition function $f : X \mapsto 2^X$ is defined by two matrices F, and H of dimensions n-by-n_e and n_H-by-n_e respectively, according to:
$$f(x) \in \Big\{F\begin{bmatrix} x & w & v & 1 \end{bmatrix}^T \mid H\begin{bmatrix} x & w & v & 1 \end{bmatrix}^T = 0, (w,v) \in [-1,1]^{n_w} \times \{-1,1\}^{n_v} \Big\}. \quad (3)$$
3) The set of initial conditions is defined via either of the following:
   a) If X_0 is finite with a small cardinality, then it can be conveniently specified by extension. We see in Section IV that per each element of X_0, one constraint needs to be included in the set of constraints of an optimization problem associated with the verification task.
   b) If X_0 is not finite, or $|X_0|$ is too large, an abstraction of X_0 can be specified by a matrix $H_0 \in \mathbb{R}^{n_{H_0} \times n_e}$ which defines a union of compact polytopes in the following way:
$$X_0 = \Big\{x \in X \mid H_0\begin{bmatrix} x & w & v & 1 \end{bmatrix}^T = 0, (w,v) \in [-1,1]^{n_w} \times \{-1,1\}^{n_v} \Big\}.$$
4) The set of terminal states X_∞ is defined by
$$X_\infty = \Big\{x \in X \mid H\begin{bmatrix} x & w & v & 1 \end{bmatrix}^T \neq 0, \forall (w,v) \in [-1,1]^{n_w} \times \{-1,1\}^{n_v} \Big\}.$$

Therefore, $\mathcal{S}(X, f, X_0, X_\infty)$ is well defined. A compact description of a MILM of a program is either of the form $\mathcal{S}(F, H, H_0, n, n_w, n_v)$, or of the form $\mathcal{S}(F, H, X_0, n, n_w, n_v)$. The MILMs can represent a broad range of computer programs of interest in control applications, including but not limited to control programs of gain scheduled linear systems in embedded applications. In addition, generalization of the model to programs with piecewise affine dynamics subject to quadratic constraints is straightforward.

Example 2: A MILM of an abstraction of the Integer Division program (Program 1: Section II-A), with all the integer variables replaced with real variables, is given by $\mathcal{S}(F, H, H_0, 4, 3, 0)$, where the matrices F, H, H_0 are displayed at the bottom of this page. Here, M is a scaling parameter used for bringing all the variables within $[-1,1]$.

2) Graph Model: Practical considerations such as universality and strong resemblance to the natural flow of computer code render graph models an attractive and convenient model for software analysis. A graph model is defined on a directed graph $G(\mathcal{N}, \mathcal{E})$ with the following elements:

1) A set of nodes $\mathcal{N} = \{\emptyset\} \cup \{1, \ldots, m\} \cup \{\bowtie\}$. These can be thought of as line numbers or code locations. Nodes $\emptyset$ and $\bowtie$ are starting and terminal nodes, respectively. The only possible transition from node $\bowtie$ is the identity transition to node $\bowtie$.
2) A set of edges $\mathcal{E} = \{(i,j,k) \mid i \in \mathcal{N},\ j \in \mathcal{O}(i)\}$, where the *outgoing set* $\mathcal{O}(i)$ is the set of all nodes to which transition from node i is possible in one step. Definition of the *incoming set* $\mathcal{I}(i)$ is analogous. The third element in the triplet (i,j,k) is the index for the kth edge between i and j, and $\mathcal{A}_{ji} = \{k \mid (i,j,k) \in \mathcal{E}\}$.
3) A set of program variables $x_l \in \Omega \subseteq \mathbb{R}$, $l \in \mathbb{Z}(1,n)$. Given $\mathcal{N}$ and n, the state space of a graph model is $X = \mathcal{N} \times \Omega^n$. The state $\widetilde{x} = (i,x)$ of a graph model has therefore, two components: The discrete component $i \in \mathcal{N}$, and the continuous component $x \in \Omega^n \subseteq \mathbb{R}^n$.
4) A set of *transition* labels $\overline{T}^k_{ji}$ assigned to every edge $(i,j,k) \in \mathcal{E}$, where $\overline{T}^k_{ji}$ maps x to the set $\overline{T}^k_{ji}x = \{T^k_{ji}(x,w,v) \mid (x,w,v) \in S^k_{ji}\}$, where $(w,v) \in [-1,1]^{n_w} \times \{-1,1\}^{n_v}$, and $T^k_{ji} : \mathbb{R}^{n+n_w+n_v} \mapsto \mathbb{R}^n$ is a polynomial function and S^k_{ji} is a semialgebraic set. If $\overline{T}^k_{ji}$ is a deterministic map, we drop S^k_{ji} and define $\overline{T}^k_{ji} \equiv T^k_{ji}(x)$.
5) A set of *predicate* labels Π^k_{ji} assigned to every edge $(i,j,k) \in \mathcal{E}$, where Π^k_{ji} is a semialgebraic set. A state

$$H_0 = \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 & 1 & 0 & 1 \\ -2 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, H = \begin{bmatrix} 0 & 2 & 0 & -2 & 1 & 0 & 0 & 1 \\ 0 & -2 & 0 & 0 & 0 & 1 & 0 & 1 \\ -2 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, F = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1/M \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

transition along the edge (i,j,k) is possible if and only if $x \in \Pi_{ji}^k$.

6) Semialgebraic invariant sets $X_i \subseteq \Omega^n,\ i \in \mathcal{N}$ are assigned to every node on the graph, such that $(i,x) \in \{(i,y)|y \in X_i\}$. Equivalently, a state $\widetilde{x} = (i,x)$ satisfying $x \in X \backslash X_i$ is unreachable.

Therefore, a graph model is a well-defined specific case of the generic model $\mathcal{S}(X,f,X_0,X_\infty)$, with $X = \mathcal{N} \times \Omega^n$, $X_0 = \{\emptyset\} \times X_\emptyset$, $X_\infty = \{\bowtie\} \times X_\bowtie$ and $f : X \mapsto 2^X$ defined as:

$$f(\widetilde{x}) \equiv f(i,x) = \left\{ (j, \overline{T}_{ji}^k x) \mid j \in \mathcal{O}(i),\ x \in \Pi_{ji}^k \cap X_i \right\}.$$

Remark 1: A few remarks are due regarding graph models:

1) The invariant set of node $\emptyset$ contains all the available information about the initial conditions of the program variables: $(\emptyset, x) \in \{(\emptyset, y)|y \in X_\emptyset\}$.
2) Multiple edges between nodes enable modeling of logical "or" or "xor" type conditional transitions. This also allows systems with nondeterministic discrete transitions to be modeled.
3) The transition label $\overline{T}_{ji}^k$ may represent a simple update rule which depends on the real-time input. For instance, if $T = Ax + Bw$, and $S = \mathbb{R}^n \times [-1,1]$, then $x \overset{\overline{T}}{\mapsto} \{Ax + Bw \mid w \in [-1,1]\}$. In general, $\overline{T}_{ji}^k$ may represent an abstraction of a nonlinearity.

Conceptually similar models, namely control flow graphs, have been reported in [4] (and the references therein) for software verification, and in [68], [69] for modeling and verification of hybrid systems. Interested readers may consult [64] for further details regarding treatment of graph models with time-varying state-dependent transitions labels which arise in modeling operations with arrays.

3) Mixed-Integer Linear over Graph Hybrid Model (MIL-GHM): The MIL-GHMs are graph models in which the effects of several lines and/or *functions* of code are compactly represented via a MILM. As a result, the graphs in such models have edges (possibly self-edges) that are labeled with matrices F and H corresponding to a MILM as the transition and predicate labels. Such models combine the flexibility provided by graph models and the compactness of MILMs. An example is presented in Section V.

C. Specifications

The specification that can be verified in our framework can generically be described as unreachability and finite-time termination.

Definition 3: A Program $\mathcal{P} \equiv \mathcal{S}(X,f,X_0,X_\infty)$ is said to satisfy the unreachability property with respect to a subset $X_- \subset X$, if for every trajectory $\mathcal{X} \equiv x(\cdot)$ of (1), and every $t \in \mathbb{Z}_+$, $x(t)$ does not belong to X_-. A program $\mathcal{P} \equiv \mathcal{S}(X,f,X_0,X_\infty)$ is said to *terminate in finite time* if every solution $\mathcal{X} = x(\cdot)$ of (1) satisfies $x(t) \in X_\infty$ for some $t \in \mathbb{Z}_+$.

As discussed below, several critical specifications associated with runtime errors are special cases of unreachability.

1) Overflow: Absence of overflow can be characterized as a special case of unreachability by defining:

$$X_- = \left\{ x \in X \mid \left\| \alpha^{-1} x \right\|_\infty > 1,\ \alpha = \mathrm{diag}\{\alpha_i\} \right\}$$

where $\alpha_i > 0$ is the overflow limit for variable i.

2) Out-of-Bounds Array Indexing: An out-of-bounds array indexing error occurs when a variable exceeding the length of an array, references an element of the array. Assuming that x_l is the corresponding integer index and L is the array length, one must verify that x_l does not exceed L at location i, where referencing occurs. This can be accomplished by defining $X_- = \{(i,x) \in X \mid |x_l| > L\}$ over a graph model and proving that X_- is unreachable. This is also similar to "assertion checking" defined next.

3) Program Assertions: An *assertion* is a mathematical expression whose validity at a specific location in the code must be verified. It usually indicates the programmer's expectation from the behavior of the program. Using graph models, verifying an assertion of the form $x \in A_i$ at *location* i in the code can be done as follows:

$$\text{assert } x \in A_i \Rightarrow \text{ define } X_- = \{(i,x) \in X \mid x \in X \backslash A_i\}.$$

Verification of an assertion of the form $x \notin A_i$ is analogous. In particular, safety assertions for division-by-zero or taking the square root (or logarithm) of positive variables are standard assertions that must be automatically included in numerical programs (cf. Sec. III-A, Table I).

4) Program Invariants: A program invariant is a property that holds throughout the execution of the program. The property indicates that the variables reside in a subset $X_I \subset X$. Thus, any method that is used for verifying unreachability of a subset $X_- \subset X$, can be applied for verifying invariance of X_I by defining $X_- = X \backslash X_I$, and vice versa.

D. The Implications of Abstractions

It is well-known in computer science that proper abstractions preserve correctness, see, e.g., [70]. It can be verified that if an $\mathcal{A}$-representation $\overline{\mathcal{S}}(\overline{X},\overline{f},\overline{X}_0,\overline{X}_\infty)$ of a program $\mathcal{P}$ satisfies the unreachability specification with respect to a set $\overline{X}_- \subset \overline{X}$, then so does the actual program, i.e., the $\mathcal{C}$-representation $\mathcal{S}(X,f,X_0,X_\infty)$, with respect to any set $X_- \subseteq \overline{X}_-$. Moreover, given (2), FTT lends itself from the $\mathcal{A}$-representation to the $\mathcal{C}$-representation. For a formal statement and proof see [64]. Since we are not concerned with undecidability issues, and in light of the preceding discussion, in the remainder of this paper we will not differentiate between abstract and concrete representations.

III. Lyapunov Invariants as Behavior Certificates

Analogous to a Lyapunov function, a Lyapunov invariant is a real-valued function of the program variables satisfying a *difference inequality* along the execution trace.

Definition 4: A (θ,μ)-*Lyapunov invariant* for the system $\mathcal{S}(X,f,X_0,X_\infty)$ is a function $V : X \mapsto \mathbb{R}$ such that

$$V(x_+) - \theta V(x) \leq -\mu,\ \forall x \in X,\ x_+ \in f(x) : x \notin X_\infty, \tag{4}$$

where $(\theta,\mu) \in [0,\infty)^2$.

Thus, a Lyapunov invariant satisfies the inequality (4) along the trajectories of $\mathcal{S}$ until they reach a terminal state X_∞. It follows from Definition 4 that a Lyapunov invariant is not necessarily nonnegative, or bounded from below, and in general it need not be monotonically decreasing. While the zero level set of V defines an invariant set in the sense that $V(x_k) \leq 0$ implies $V(x_{k+l}) \leq 0$, for all $l \geq 0$, monotonicity depends on θ and the initial condition. For instance, if $V(x_0) \leq 0$, for all $x_0 \in X_0$, then (4) implies that $V(x) \leq 0$ along the trajectories of $\mathcal{S}$, however, $V(x)$ may not be monotonic if $\theta < 1$, though it will be monotonic for $\theta \geq 1$. Furthermore, the level sets of a Lyapunov invariant need not be bounded closed curves.

Natural Lyapunov invariants for graph models are functions of the form

$$V(\widetilde{x}) \equiv V(i,x) = \sigma_i(x), \quad i \in N, \tag{5}$$

which assign a polynomial Lyapunov function to every node $i \in \mathcal{N}$ on the graph $G(\mathcal{N},\mathcal{E})$. However, we stress that this is simply one way of assigning Lyapunov functions to graph models, and certainly not the most general, or the most powerful way.

The following Proposition formalizes the interpretation of Lyapunov invariants for the specific modeling languages.

Proposition 2: Let $\mathcal{S}(F,H,X_0,n,n_w,n_v)$ and properly labeled graph $G(\mathcal{N},\mathcal{E})$ be the MIL and graph models for a computer program $\mathcal{P}$. The function $V : [-1,1]^n \mapsto \mathbb{R}$ is a (θ,μ)-Lyapunov invariant for $\mathcal{P}$ if it satisfies:

$$V(Fx_e) - \theta V(x) \leq -\mu, \qquad \forall (x,x_e) \in [-1,1]^n \times \Xi,$$

where

$$\Xi = \Big\{ (x,w,v,1) \mid H \begin{bmatrix} x & w & v & 1 \end{bmatrix}^T = 0, \\ (w,v) \in [-1,1]^{n_w} \times \{-1,1\}^{n_v} \Big\}.$$

The function $V : \mathcal{N} \times \mathbb{R}^n \mapsto \mathbb{R}$, satisfying (5) is a (θ,μ)-Lyapunov invariant for $\mathcal{P}$ if

$$\sigma_j(x_+) - \theta \sigma_i(x) \leq -\mu, \\ \forall (i,j,k) \in \mathcal{E},\ (x,x_+) \in (X_i \cap \Pi^k_{ji}) \times \overline{T}^k_{ji} x.$$

Note that a generalization of (4) allows θ and μ to depend on the state, although simultaneous search for $\theta(x)$ and $V(x)$ leads us to non-convex conditions (in the parameters of V and θ), unless the dependence of θ on the state is fixed a-priori. We allow θ to depend on the discrete component of the state in the following way:

$$\sigma_j(x_+) - \theta^k_{ji} \sigma_i(x) \leq -\mu_{ji}, \\ \forall (i,j,k) \in \mathcal{E},\ (x,x_+) \in (X_i \cap \Pi^k_{ji}) \times \overline{T}^k_{ji} x \tag{6}$$

A. Behavior Certificates

1) Finite-Time Termination (FTT) Certificates: The following proposition is applicable to FTT analysis of both finite and infinite state models.

Proposition 3: **Finite-Time Termination.** Consider a program $\mathcal{P}$, and its dynamical system model $\mathcal{S}(X,f,X_0,X_\infty)$. If there exists a (θ,μ)-Lyapunov invariant $V : X \mapsto \mathbb{R}$, uniformly bounded on $X \backslash X_\infty$, satisfying (4) and the following conditions

$$V(x) \leq -\eta \leq 0, \qquad \forall x \in X_0 \tag{7}$$

$$\mu + (\theta - 1)\|V\|_\infty > 0 \tag{8}$$

$$\max(\mu,\eta) > 0 \tag{9}$$

where $\|V\|_\infty = \sup_{x \in X \backslash X_\infty} |V(x)| < \infty$, then $\mathcal{P}$ terminates in finite time, and an upper-bound on the number of iterations is given by

$$T_u = \begin{cases} \dfrac{\log(\mu + (\theta-1)\|V\|_\infty) - \log(\mu)}{\log\theta}, & \theta \neq 1,\ \mu > 0 \\ \dfrac{\log(\|V\|_\infty) - \log(\eta)}{\log\theta}, & \theta \neq 1,\ \mu = 0 \\ \|V\|_\infty / \mu, & \theta = 1 \end{cases} \tag{10}$$

Proof: The proof is omitted but can be found in [64]. ■

Example 3: Consider the Integer Division Program of Example 1. The function $V : X \mapsto \mathbb{R}$, defined according to $V : (\mathrm{dd},\mathrm{dr},\mathrm{q},\mathrm{r}) \mapsto \mathrm{r}$ is a $(1,\mathrm{dr})$-Lyapunov invariant for Integer Division: at every step, V decreases by $\mathrm{dr} > 0$. Since X is finite, Program 1 terminates in finite time. This, however, only proves absence of infinite loops. The program could terminate with an overflow.

Remark 2: Parallels of these ideas for proving termination of programs exist in the classical computer science literature under the notion of *ranking functions*. A ranking function is a map $g : X \mapsto Y$, such that $(Y,<)$ forms a well ordered set, i.e., every subset of Y has a smallest element. If a program is defined by a relation $R \subset X \times X$, and $g : X \mapsto Y$ is a ranking function satisfying $g(x) < g(x')$ for all $(x,x') \in R$, then g is a certificate for termination of the program. Recently, construction and verification of ranking functions based on linear optimization, or binary reachability analysis have gained attention and shown success in verification of device drivers of the Windows operating system [71], [72].

2) Separating Manifolds and Certificates of Boundedness: Let V be a Lyapunov invariant satisfying (4) with $\theta = 1$. The level sets of V, defined by $\mathcal{L}_r(V) \stackrel{\text{def}}{=} \{x \in X : V(x) < r\}$, are invariant with respect to (1) in the sense that $x(t+1) \in \mathcal{L}_r(V)$ whenever $x(t) \in \mathcal{L}_r(V)$. However, for $r = 0$, the level sets $\mathcal{L}_r(V)$ remain invariant with respect to (1) for any $\theta \geq 0$. This is an important property with the implication that $\theta = 1$ (i.e., monotonicity) is not necessary for establishing a separating manifold between the reachable set and the unsafe regions of the state space (cf. Theorem 1).

Theorem 1: **Lyapunov Invariants as Separating Manifolds.** Let $\mathcal{V}$ denote the set of all (θ,μ)-Lyapunov invariants satisfying (4) for program $\mathcal{P} \equiv \mathcal{S}(X,f,X_0,X_\infty)$. Let I be the identity map, and for $h \in \{f, I\}$ define

$$h^{-1}(X_-) = \{x \in X \mid h(x) \cap X_- \neq \varnothing\}.$$

A subset $X_- \subset X$, where $X_- \cap X_0 = \varnothing$ can never be reached

along the trajectories of $\mathcal{P}$, if there exists $V \in \mathcal{V}$ satisfying

$$\sup_{x \in X_0} V(x) \; < \; \inf_{x \in h^{-1}(X_-)} V(x) \tag{11}$$

and either $\theta = 1$, or one of the following two conditions hold:

$$\text{(I)} \;\; \theta < 1 \qquad \text{and} \qquad \inf_{x \in h^{-1}(X_-)} V(x) > 0. \tag{12}$$

$$\text{(II)} \;\; \theta > 1 \qquad \text{and} \qquad \sup_{x \in X_0} V(x) \leq 0. \tag{13}$$

Proof: The proof is presented in Appendix II. ∎

The following corollary follows from Theorem 1 and Proposition 3, and presents computationally implementable criteria (cf. SectionIV) for simultaneously establishing FTT and absence of overflow.

Corollary 1: **Overflow and FTT Analysis** Consider a program $\mathcal{P}$, and its dynamical system model $\mathcal{S}(X, f, X_0, X_\infty)$. Let $\alpha > 0$ be a diagonal matrix specifying the overflow limit, and let $X_- = \{x \in X \mid \left\|\alpha^{-1} x\right\|_\infty > 1\}$. Let $q \in \mathbb{N} \cup \{\infty\}$, $h \in \{f, I\}$, and let the function $V : X \mapsto \mathbb{R}$ be a (θ, μ)-Lyapunov invariant for $\mathcal{S}$ satisfying

$$V(x) \leq 0 \qquad \forall x \in X_0. \tag{14}$$

$$V(x) \geq \sup \left\{ \left\|\alpha^{-1} h(x)\right\|_q - 1 \right\} \qquad \forall x \in X. \tag{15}$$

Then, an *overflow runtime error* will not occur during any execution of $\mathcal{P}$. In addition, if $\mu > 0$ and $\mu + \theta > 1$, then, $\mathcal{P}$ terminates in at most T_u iterations where $T_u = \mu^{-1}$ if $\theta = 1$, and for $\theta \neq 1$ we have:

$$\begin{aligned} T_u &= \frac{\log \;\; (\mu + (\theta - 1) \left\|V\right\|_\infty) - \log \mu}{\log \theta} \\ &\leq \frac{\log (\mu + \theta - 1) - \log \mu}{\log \theta} \end{aligned} \tag{16}$$

where, $\left\|V\right\|_\infty = \sup_{x \in X \setminus \{X_- \cup X_\infty\}} |V(x)|$.

Proof: The proof is presented in Appendix II. ∎

Application of Corollary 1 with $h = f$ typically leads to less conservative results compared with $h = I$, though the computational costs are also higher. See [64] for remarks on variations of Corollary 1 to trade off conservativeness and computational complexity.

a) General Unreachability and FTT Analysis over Graph Models: The results presented so far in this section (Theorem 1, Corollary 1, and Proposition 3) are readily applicable to MILMs. These results are applied in Section IV to formulate the verification problem as a convex optimization problem. Herein, we present an adaptation of these results to analysis of graph models.

Definition 5: A cycle $\mathcal{C}_m$ on a graph $G(\mathcal{N}, \mathcal{E})$ is an ordered list of m triplets (n_1, n_2, k_1), (n_2, n_3, k_2), ..., (n_m, n_{m+1}, k_m), where $n_{m+1} = n_1$, and $(n_j, n_{j+1}, k_j) \in \mathcal{E}$, $\forall j \in \mathbb{Z}(1, m)$. A simple cycle is a cycle with no strict subcycles.

Corollary 2: **Unreachability and FTT Analysis of Graph Models**. Consider a program $\mathcal{P}$ and its graph model $G(\mathcal{N}, \mathcal{E})$. Let $V(i, x) = \sigma_i(x)$ be a Lyapunov invariant for $G(\mathcal{N}, \mathcal{E})$, satisfying (6) and

$$\sigma_\emptyset(x) \leq 0, \qquad \forall x \in X_\emptyset \tag{17}$$

and either one of the following two conditions:

$$\text{(I)} : \sigma_i(x) > 0, \qquad \forall x \in X_i \cap X_{i-}, \; i \in \mathcal{N} \setminus \{\emptyset\} \tag{18}$$

$$\text{(II)} : \sigma_i(x) > 0, \qquad \forall x \in X_j \cap T^{-1}(X_{i-}), \; i \in \mathcal{N} \setminus \{\emptyset\}, \; j \in \mathcal{I}(i), \; T \in \left\{ \overline{T}_{ij}^k \right\} \tag{19}$$

where

$$T^{-1}(X_{i-}) = \{x \in X_i | T(x) \cap X_{i-} \neq \varnothing\}.$$

Then, $\mathcal{P}$ satisfies the unreachability property w.r.t. the collection of sets X_{i-}, $i \in \mathcal{N} \setminus \{\emptyset\}$. In addition, if for every simple cycle $\mathcal{C} \in G$, the following three conditions hold:

$$(\theta(\mathcal{C}) - 1) \left\|\sigma(\mathcal{C})\right\|_\infty + \mu(\mathcal{C}) > 0, \tag{20a}$$

$$\mu(\mathcal{C}) > 0, \tag{20b}$$

$$\left\|\sigma(\mathcal{C})\right\|_\infty < \infty, \tag{20c}$$

where

$$\theta(\mathcal{C}) = \prod_{(i,j,k) \in \mathcal{C}} \theta_{ij}^k, \qquad \mu(\mathcal{C}) = \max_{(i,j,k) \in \mathcal{C}} \mu_{ij}^k,$$

$$\left\|\sigma(\mathcal{C})\right\|_\infty = \max_{(i,.,.) \in \mathcal{C}} \; \sup_{x \in X_i \setminus X_{i-}} |\sigma_i(x)|$$

then $\mathcal{P}$ terminates in at most T_u iterations where

$$T_u = \sum_{\mathcal{C} \in G : \theta(\mathcal{C}) = 1} \frac{\left\|\sigma(\mathcal{C})\right\|_\infty}{\mu(\mathcal{C})} + \sum_{\mathcal{C} \in G : \theta(\mathcal{C}) \neq 1} \frac{\log \;\; ((\theta(\mathcal{C}) - 1) \left\|\sigma(\mathcal{C})\right\|_\infty + \mu(\mathcal{C})) - \log \;\; \mu(\mathcal{C})}{\log \theta(\mathcal{C})}$$

Proof: The proof is presented in Appendix II. ∎

For verification against an overflow violation specified by a diagonal matrix $\alpha > 0$, Corollary 2 is applied with $X_- = \{x \in \mathbb{R}^n \mid \left\|\alpha^{-1} x\right\|_\infty > 1\}$. Hence, (18) becomes

$$\sigma_i(x) \geq p(x) (\left\|\alpha^{-1} x\right\|_q - 1), \quad \forall x \in X_i, \; i \in \mathcal{N} \setminus \{\emptyset\},$$

where $p(x) > 0$. User-specified assertions, as well as many standard safety specifications, such as absence of division-by-zero can be verified using Corollary 2 (See Table I).

– Identification of Dead Code: Suppose that we wish to verify that a discrete location $i \in \mathcal{N} \setminus \{\emptyset\}$ in a graph model $G(\mathcal{N}, \mathcal{E})$ is unreachable. If a function satisfying the criteria of Corollary 2 with $X_{i-} = \mathbb{R}^n$ can be found, then location i can never be reached. Condition (18) then becomes $\sigma_i(x) \geq 0$, for all $x \in \mathbb{R}^n$.

TABLE I
APPLICATION OF COROLLARY 2 TO VERIFICATION OF VARIOUS SAFETY SPECIFICATIONS.

			apply Corollary 2 with:
loc i:	assert $x \in X_a$	$\rightarrow$	$X_{i-} = \{x \in \mathbb{R}^n \mid x \notin X_a\}$
loc i:	assert $x \notin X_a$	$\rightarrow$	$X_{i-} = \{x \in \mathbb{R}^n \mid x \in X_a\}$
loc i:	(expression)/x_o	$\rightarrow$	$X_{i-} = \{x \in \mathbb{R}^n \mid x_o = 0\}$
loc i:	$\sqrt[2k]{x_o}$	$\rightarrow$	$X_{i-} = \{x \in \mathbb{R}^n \mid x_o < 0\}$
loc i:	$\log(x_o)$	$\rightarrow$	$X_{i-} = \{x \in \mathbb{R}^n \mid x_o \leq 0\}$
loc i:	dead code	$\rightarrow$	$X_{i-} = \mathbb{R}^n$

Example 4: Consider Program 3 and its graph model as displayed in Figure 1.

```
        void ComputeTurnRate (void)
L0 :    {double x = {0}; double y = {*PtrToY};
L1 :    while (1)
L2 :    {     y = *PtrToY;
L3 :          x = (5 * sin(y) + 1)/3;
L4 :          if x > -1 {
L5 :                x = x + 1.0472;
L6 :                TurnRate = y/x; }
L7 :          else {
L8 :                TurnRate = 100 * y/3.1416 }}
```

Program 3

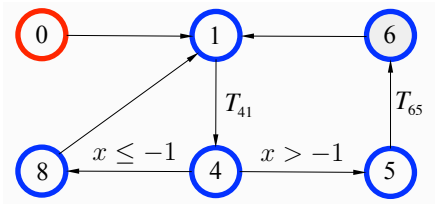

Fig. 1. Graph Model of an Abstraction of Program 3

Note that x can be zero right after the assignment at L3 : $\mathrm{x} = (5\sin(\mathrm{y}) + 1)/3$. However, at location L6, x cannot be zero and division-by-zero will not occur. The graph model of an abstraction of Program 3 is defined by the following elements: $T_{65} : \mathrm{x} \mapsto \mathrm{x} + 1.0472$, and $T_{41} : \mathrm{x} \mapsto [-4/3, 2]$. The other transition labels are identity. The only non-universal predicate labels are Π_{54} and Π_{84} as shown in the figure. Define

$$\sigma_6(\mathrm{x}) = -\mathrm{x}^2 - 100\mathrm{x} + 1,$$
$$\sigma_5(\mathrm{x}) = -(\mathrm{x} + 1309/1250)^2 - 100\mathrm{x} - 2543/25$$
$$\sigma_0(\mathrm{x}) = \sigma_1(\mathrm{x}) = \sigma_4(\mathrm{x}) = \sigma_8(\mathrm{x}) = -\mathrm{x}^2 + 2\mathrm{x} - 3.$$

It can be verified that $V(\mathrm{x}) = \sigma_i(\mathrm{x})$ is a $(\theta, 1)$-Lyapunov invariant for Program 3 with variable rates: $\theta_{65} = 1$, and $\theta_{ij} = 0, \forall (i,j) \neq (6,5)$. Since

$$-2 = \sup_{\mathrm{x} \in X_0} \sigma_0(\mathrm{x}) < \inf_{\mathrm{x} \in X_-} \sigma_6(\mathrm{x}) = 1$$

the state $(6, \mathrm{x} = 0)$ cannot be reached. Hence, a division by zero will never occur. We show in the next section how to find such functions in general.

Remark 3: It is well-known that *lifting* the state space of a dynamical system in a systematic way can result in a significant improvement in the quality of analysis using simple (e.g. quadratic) Lyapunov functions, see, e.g., [73]. Naturally, the idea of lifting extends to software analysis. As we already mentioned, assigning Lyapunov invariants to the nodes of a graph model according to (5) is one intuitive way for defining Lyapunov invariant for graph models. The overall framework, however, can accommodate more general formulations, either by allowing a more complicated dependency (for the invariants) on the discrete component of the state space, or by lifting the state space model. The latter can be done, for instance, by augmenting additional states to encode more *memory* from past operations or visited locations. Parallel of this idea exists in the computer science literature under the notion of progress invariants [74]. It is also known that Lyapunov analysis of *reduced* graph models obtained from proper projection of an original graph to a lower-dimensional graph can be computationally and methodologically advantageous [75] since several transitions can be composed into one transition and the Lyapunov inequalities are required to be satisfied only for the aggregate transition. Parallels of this idea exist in the control literature under the notion of non-monotonic Lyapunov functions [76] and in the computer science literature under the notion of relational abstractions [22].

IV. Computation of Lyapunov Invariants

It is well known that the main difficulty in using Lyapunov functions in system analysis is finding them. Naturally, using Lyapunov invariants in software analysis inherits the same difficulties. However, the recent advances in hardware and software technology, e.g., semi-definite programming [77], [78], and linear programming software [79] present an opportunity for new approaches to software verification based on numerical optimization.

A. Preliminaries

1) Convex Parameterization of Lyapunov Invariants: The chances of finding a Lyapunov invariant are increased when (4) is only required on a subset of $X \backslash X_\infty$. For instance, for $\theta \leq 1$, it is tempting to replace (4) with

$$V(x_+) - \theta V(x) \leq -\mu, \quad \forall x \in X \backslash X_\infty : V(x) < 1,\ x_+ \in f(x) \qquad (21)$$

In this formulation V is not required to satisfy (4) for those states which cannot be reached from X_0. However, the set of functions $V : X \mapsto \mathbb{R}$ satisfying (21) is not convex and finding a solution for (21) is typically much harder than (4). Such non-convex formulations are not considered in this paper.

The first step in the search for a function $V : X \mapsto \mathbb{R}$ satisfying (4) is selecting a finite-dimensional linear parameterization of a candidate function V:

$$V(x) = V_\tau(x) = \sum_{k=1}^{n} \tau_k V_k(x),\ \tau = (\tau_k)_{k=1}^n,\ \tau_k \in \mathbb{R}, \qquad (22)$$

where $V_k : X \mapsto \mathbb{R}$ are fixed basis functions. Next, for every $\tau = (\tau_k)_{k=1}^N$ let

$$\phi(\tau) = \max_{x \in X \backslash X_\infty,\ x_+ \in f(x)} V_\tau(x_+) - \theta V_\tau(x),$$

(assuming for simplicity that the maximum does exist). Since $\phi(\cdot)$ is a maximum of a family of linear functions in τ, $\phi(\cdot)$ is a convex function. If minimizing $\phi(\cdot)$ over the unit disk yields a negative minimum, the optimal τ^* defines a valid Lyapunov invariant $V_{\tau^*}(x)$. Otherwise, no linear combination (22) yields a valid solution for (4).

The success and efficiency of the proposed approach depend on computability of $\phi(\cdot)$ and its subgradients. While $\phi(\cdot)$ is convex, the same does not necessarily hold for $V_\tau(x_+) - \theta V_\tau(x)$ as a function of x. In fact, if $X \backslash X_\infty$ is non-convex, which is often the case even for very simple programs,

computation of $\phi(\cdot)$ becomes a non-convex optimization problem, even if $V_\tau(x_+) - V_\tau(x)$ is a nice (e.g. linear or concave and smooth) function of x. To get around this hurdle, we propose using convex relaxation techniques which essentially lead to computation of a convex upper bound for $\phi(\tau)$.

2) Convex Relaxation Techniques: Such techniques constitute a broad class of methods for constructing finite-dimensional, convex approximations for non-convex optimization problems. Some of the results most relevant to the software verification framework presented in this paper can be found in [80] for SDP relaxation of binary integer programs, [81] and [82] for SDP relaxation of quadratic programs, [83] for $\mathcal{S}$-Procedure in robustness analysis, and [84], [63] for sum-of-squares relaxation in polynomial non-negativity verification. We provide a brief overview of the latter two techniques.

a) The $\mathcal{S}$-Procedure : The $\mathcal{S}$-Procedure is commonly used for construction of Lyapunov functions for nonlinear dynamical systems. Let functions $\phi_i : X \mapsto \mathbb{R}$, $i \in \mathbb{Z}(0, m)$, and $\psi_j : X \mapsto \mathbb{R}$, $j \in \mathbb{Z}(1, n)$ be given, and suppose that we are concerned with evaluating the following assertions:

(I): $\phi_0(x) > 0,\ \forall x \in \{x \in X \mid \phi_i(x) \geq 0,\ \psi_j(x) = 0,\ \forall i, j\}$

(II): $\exists \tau_i \in \mathbb{R}^+,\ \exists \mu_j \in \mathbb{R},$ such that

$$\phi_0(x) > \sum_{i=1}^{m} \tau_i \phi_i(x) + \sum_{j=1}^{n} \mu_j \psi_j(x).$$

The implication (II) $\rightarrow$ (I) is trivial. The process of replacing assertion (I) by its *relaxed* version (II) is called the $\mathcal{S}$-Procedure. Note that (II) is convex in decision variables τ_i and μ_j. The implication (I) $\rightarrow$ (II) is generally not true and the $\mathcal{S}$-Procedure is called lossless for special cases where (I) and (II) are equivalent. A well-known such case is when $m = 1$, $n = 0$, and ϕ_0, ϕ_1 are quadratic functionals. A comprehensive discussion of the $\mathcal{S}$-Procedure as well as available results on its losslessness can be found in [85]. Other variations of the $\mathcal{S}$-Procedure with non-strict inequalities exist as well.

b) Sum-of-Squares (SOS) Relaxation : The SOS relaxation technique can be interpreted as the generalized version of the $\mathcal{S}$-Procedure and is concerned with verification of the following assertion:

$$f_j(x) \geq 0,\ \forall j \in J,\ g_k(x) \neq 0,\ \forall k \in K,\ h_l(x) = 0,\ \forall l \in L \Rightarrow -f_0(x) \geq 0, \quad (23)$$

where f_j, g_k, h_l are polynomial functions. It is easy to see that the problem is equivalent to verification of emptiness of a semialgebraic set, a necessary and sufficient condition for which is given by the Positivstellensatz Theorem [86]. In practice, sufficient conditions in the form of nonnegativity of polynomials are formulated, which are in turn relaxed to SOS conditions. Let $\Sigma[y_1, \ldots, y_m]$ denote the set of SOS polynomials in m variables $y_1, ..., y_m$, i.e. the set of polynomials that can be represented as $p = \sum_{i=1}^{t} p_i^2,\ p_i \in \mathbb{P}_m$, where $\mathbb{P}_m$ is the polynomial ring of m variables with real coefficients. Then, a sufficient condition for (23) is that there exist SOS polynomials $\tau_0, \tau_i, \tau_{ij} \in \Sigma[x]$ and polynomials ρ_l, such that

$$\tau_0 + \sum\nolimits_i \tau_i f_i + \sum\nolimits_{i,j} \tau_{ij} f_i f_j + \sum\nolimits_l \rho_l h_l + (\prod g_k)^2 = 0$$

Matlab toolboxes such as SOSTOOLS [87], YALMIP [88], SparsePOP [89], and GloptiPoly [90] automate the process of converting an SOS problem to an SDP, which is subsequently solved by available software packages such as LMILAB [77], or SeDuMi [78]. Interested readers are referred to [63], [91], [84], [92], [93], [62], [61] for more details.

Before we proceed, we introduce the following notation. A linear parameterization of the subspace of polynomial functionals with total degree not greater than d is given by:

$$\mathcal{V}_x^d = \left\{ V : \mathbb{R}^n \mapsto \mathbb{R} \mid V(x) = K^T Z(x),\ K \in \mathbb{R}^{\binom{n+d}{d}} \right\},$$

where $Z(x)$ is a vector of length $\binom{n+d}{d}$, consisting of all monomials of degree less than or equal to d in n variables $x_1, ..., x_n$. In particular, the linear parameterization of the space of quadratic functionals mapping $\mathbb{R}^n$ to $\mathbb{R}$ is given by:

$$\mathcal{V}_x^2 = \left\{ V : \mathbb{R}^n \mapsto \mathbb{R} \mid V(x) = \begin{bmatrix} x \\ 1 \end{bmatrix}^T P \begin{bmatrix} x \\ 1 \end{bmatrix},\ P \in \mathbb{S}^{n+1} \right\},$$

where $\mathbb{S}^n$ is the set of n-by-n symmetric matrices.

B. Optimization of Lyapunov Invariants for MILMs

Natural Lyapunov invariant candidates for MILMs are quadratic and affine functionals. Given a program $\mathcal{P}$ and its MIL model $\mathcal{S}(F, H, X_0, n, n_w, n_v)$, for convenience in notation, we define matrices $L_i,\ i = 1, \ldots, 5$ as follows:

$$L_1 = \begin{bmatrix} F \\ L_5 \end{bmatrix},\ L_2 = \begin{bmatrix} I_n & 0_{n\times(n_e-n)} \\ 0_{1\times(n_e-1)} & 1 \end{bmatrix},$$

$$L_3 = \begin{bmatrix} I_{n+n_w} \\ 0_{(n_v+1)\times(n+n_w)} \end{bmatrix}^T,\ L_4 = \begin{bmatrix} 0_{(n+n_w)\times n_v} \\ I_{n_v} \\ 0_{1\times n_v} \end{bmatrix}^T,$$

$$L_5 = \begin{bmatrix} 0_{(n_e-1)\times 1} \\ 1 \end{bmatrix}^T.$$

1) Quadratic Invariants: We have the following proposition.

Proposition 4: **SDP Search for Quadratic Invariants.** A program $\mathcal{P} \equiv \mathcal{S}(F, H, X_0, n, n_w, n_v)$ admits a quadratic (θ, μ)-Lyapunov invariant $V \in \mathcal{V}_x^2$, if there exists a matrix $Y \in \mathbb{R}^{n_e \times n_H}$, $n_e = n + n_w + n_v + 1$, diagonal matrices $D_v \in \mathbb{D}^{n_v}$, and $D_{xw} \in \mathbb{D}_+^{n+n_w}$, and a symmetric matrix $P \in \mathbb{S}^{n+1}$, satisfying the following LMI:

$$L_1^T P L_1 - \theta L_2^T P L_2 \preceq \mathrm{He}(YH) + L_3^T D_{xw} L_3 + L_4^T D_v L_4 - (\lambda + \mu) L_5^T L_5$$

where $\lambda = \mathrm{Trace}\, D_{xw} + \mathrm{Trace}\, D_v$, and $D_{xw} \succeq 0$.

Proof: The proof is presented in Appendix II. ■

The following theorem summarizes verification of absence of overflow and/or FTT for MILMs. The result follows from Proposition 4 and Corollary 1 with $q = 2$, and $h = f$, though the theorem is presented without a detailed proof.

Theorem 2: **Optimization-Based MILM Verification.** Let $\alpha : 0 \prec \alpha \preceq I_n$ be a diagonal positive definite matrix specifying the overflow limit. An overflow runtime error does not occur during any execution of $\mathcal{P}$ if there exist matrices $Y_i \in \mathbb{R}^{n_e \times n_H}$, diagonal matrices $D_{iv} \in \mathbb{D}^{n_v}$, positive semidefinite diagonal matrices $D_{ixw} \in \mathbb{D}_+^{n+n_w}$, and a symmetric matrix $P \in \mathbb{S}^{n+1}$ satisfying the following LMIs:

$$\left[\begin{array}{cc} x_0 & 1 \end{array}\right] P \left[\begin{array}{cc} x_0 & 1 \end{array}\right]^T \leq 0, \qquad \forall x_0 \in X_0$$

$$L_1^T P L_1 - \theta L_2^T P L_2 \preceq \mathrm{He}\left(Y_1 H\right) + L_3^T D_{1xw} L_3 + L_4^T D_{1v} L_4 - \left(\lambda_1 + \mu\right) L_5^T L_5$$

$$L_1^T \Lambda L_1 - L_2^T P L_2 \preceq \mathrm{He}\left(Y_2 H\right) + L_3^T D_{2xw} L_3 + L_4^T D_{2v} L_4 - \lambda_2 L_5^T L_5$$

where $\Lambda = \mathrm{diag}\left\{\alpha^{-2}, -1\right\}$, $\lambda_i = \mathrm{Trace}\, D_{ixw} + \mathrm{Trace}\, D_{iv}$, and $D_{ixw} \succeq 0$, $i = 1, 2$. In addition, if $\mu + \theta > 1$, then $\mathcal{P}$ terminates in a most T_u steps where T_u is given in (16).

2) Affine Invariants: Affine Lyapunov invariants can often establish strong properties, e.g., boundedness, for variables with simple uncoupled dynamics (e.g. counters) at a low computational cost. For variables with more complicated dynamics, affine invariants may simply establish sign-invariance or more generally, upper or lower bounds on a linear combination of certain variables. We will see in Section V that establishing these simple behavioral properties is important as they can be recursively added to the model (e.g., the matrix H in a MILM, or the invariant sets X_i in a graph model) to improve the chances of success in proving stronger properties via higher-order invariants. It is possible to search for the affine invariants via semidefinite programming or linear programming.

Proposition 5: **SDP Formulation of Linear Invariants.** There exists a (θ, μ)-Lyapunov invariant $V \in \mathcal{V}_x^1$ for a program $\mathcal{P} \equiv \mathcal{S}\left(F, H, X_0, n, n_w, n_v\right)$, if there exists a matrix $Y \in \mathbb{R}^{n_e \times n_H}$, a diagonal matrix $D_v \in \mathbb{D}^{n_v}$, a positive semidefinite diagonal matrix $D_{xw} \in \mathbb{D}_+^{(n+n_w)\times(n+n_w)}$, and a matrix $K \in \mathbb{R}^{n+1}$ satisfying the following LMI:

$$\mathrm{He}(L_1^T K L_5 - \theta L_5^T K^T L_2) \prec \mathrm{He}(Y H) + L_3^T D_{xw} L_3 + L_4^T D_v L_4 - \left(\lambda + \mu\right) L_5^T L_5$$

where $\lambda = \mathrm{Trace}\, D_{xw} + \mathrm{Trace}\, D_v$ and $0 \preceq D_{xw}$.

Proposition 6: **LP Formulation of Linear Invariants.** There exists a (θ, μ)-Lyapunov invariant for a program $\mathcal{P} \equiv \mathcal{S}\left(F, H, X_0, n, n_w, n_v\right)$ in the class $\mathcal{V}_x^1$, if there exists a matrix $Y \in \mathbb{R}^{1 \times n_H}$, and nonnegative matrices $\underline{D}_v$, $\overline{D}_v \in \mathbb{R}^{1 \times n_v}$, $\underline{D}_{xw}$, $\overline{D}_{xw} \in \mathbb{R}^{1 \times (n+n_w)}$, and a matrix $K \in \mathbb{R}^{n+1}$ satisfying:

$$K^T L_1 - \theta K^T L_2 - Y H - (\underline{D}_{xw} - \overline{D}_{xw}) L_3 - (\underline{D}_v - \overline{D}_v) L_4 - \left(D_1 + \mu\right) L_5 = 0$$

$$D_1 + \left(\overline{D}_v + \underline{D}_v\right) \mathbf{1}_r + \left(\overline{D}_{xw} + \underline{D}_{xw}\right) \mathbf{1}_{n+n_w} \leq 0$$

$$\overline{D}_v,\ \underline{D}_v,\ \overline{D}_{xw},\ \underline{D}_{xw} \geq 0$$

where D_1 is either 0 or -1.

The advantage of using SDP for computation of linear invariants is that efficient SDP relaxations for treatment of binary variables exists [81], [82], [94], though the computational costs are typically higher than the LP-based approaches. In contrast, linear programming relaxations of the binary constraints are more involved than the corresponding SDP relaxations. Two extreme remedies can be readily considered. The first is to relax the binary constraints and treat the variables as continuous variables. The second is to consider each of the 2^{n_v} different possibilities (one for each vertex of $\{-1, 1\}^{n_v}$) separately. This approach is practical only if n_v is small. More sophisticated schemes can be developed based on hierarchical relaxations or convex hull approximations of binary integer programs [95], [80]. The survey paper [62] also covers some recent developments in this direction based on polynomial optimization.

C. Optimization of Lyapunov Invariants for Graph Models

A linear parametrization of Lyapunov invariants for graph models is defined according to (5), where for every $i \in \mathcal{N}$, we have $\sigma_i(\cdot) \in \mathcal{V}_x^{d(i)}$, where $d(i)$ is a selected degree bound for $\sigma_i(\cdot)$. Depending on the dynamics of the model, the degree bounds $d(i)$, and the convex relaxation technique, the corresponding optimization problem becomes a linear, semidefinite, or SOS optimization problem.

1) Node-wise Polynomial Invariants: We present generic conditions for verification over graph models using SOS programming. Although LMI conditions for verification of *linear graph models* using quadratic invariants and the $\mathcal{S}$-Procedure for relaxation of non-convex constraints can be formulated, we do not present them here due to space limitations. Such formulations are presented in the extended report [64], along with executable Matlab code in [96]. The following theorem follows from Corollary 2.

Theorem 3: **Optimization-Based Graph Model Verification.** Consider a program $\mathcal{P}$, and its graph model $G(\mathcal{N}, \mathcal{E})$. Let $V : \Omega^n \mapsto \mathbb{R}$, be given by (5), where $\sigma_i(\cdot) \in \mathcal{V}_x^{d(i)}$. Then, the functions $\sigma_i(\cdot)$, $i \in \mathcal{N}$ define a Lyapunov invariant for $\mathcal{P}$, if for all $(i, j, k) \in \mathcal{E}$ we have:

$$-\sigma_j(T_{ji}^k(x, w)) + \theta_{ji}^k \sigma_i(x) - \mu_{ji}^k \in \Sigma[x, w], \quad \text{subject to } (x, w) \in \left(\left(X_i \cap \Pi_{ji}^k\right) \times [-1, 1]^{n_w}\right) \cap S_{ji}^k \quad (24)$$

Furthermore, $\mathcal{P}$ satisfies the unreachability property w.r.t. the collection of sets X_{i-}, $i \in \mathcal{N} \backslash \{\emptyset\}$, if there exist $\varepsilon_i \in (0, \infty)$, $i \in \mathcal{N} \backslash \{\emptyset\}$, such that

$$-\sigma_\emptyset(x) \in \Sigma[x] \ \text{ subject to } x \in X_\emptyset \quad (25)$$

$$\sigma_i(x) - \varepsilon_i \in \Sigma[x] \ \text{ subject to } x \in X_i \cap X_{i-},\ i \in \mathcal{N} \backslash \{\emptyset\} \quad (26)$$

As discussed in Section IV-A2b, the SOS relaxation techniques can be applied for formulating the search problem for functions σ_i satisfying (24)–(26) as a convex optimization problem. For instance, if

$$\left(\left(X_i \cap \Pi_{ji}^k\right) \times [-1, 1]^{n_w}\right) \cap S_{ji}^k = \left\{(x, w) \mid f_p(x, w) \geq 0,\ h_l(x, w) = 0\right\},$$

then, (24) can be relaxed as an SOS optimization problem of the following form:

$$-\sigma_j(T^k_{ji}(x,w)) + \theta^k_{ji}\sigma_i(x) - \mu^k_{ji} - \sum_p \tau_p f_p -$$
$$\sum_{p,q} \tau_{pq} f_p f_q - \sum_l \rho_l h_l \in \Sigma[x,w], \text{ s.t. } \tau_p, \tau_{pq} \in \Sigma[x,w].$$

The SOS optimization problems can then be formulated as semidefinite programs using existing software packages [87], [88], [89], [90].

V. Case Study

In this section we apply the framework to analysis of Program 4, displayed below.

```
        /* EuclideanDivision.c */
F0 :    int IntegerDivision ( int dd, int dr )
F1 :    {int q = {0}; int r = {dd};
F2 :    while (r >= dr) {
F3 :        q = q + 1;
F4 :        r = r - dr;
F⋈:     return r; }
L0 :    int main ( int X, int Y ) {
L1 :    int rem = {0};
L2 :    while (Y > 0) {
L3 :        rem = IntegerDivision (X , Y);
L4 :        X = Y;
L5 :        Y = rem;
L⋈:     return X; }}
```

Program 4: Euclidean Division Algorithm

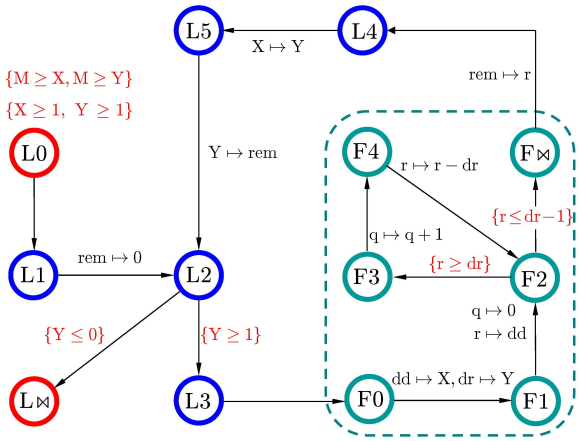

Fig. 2. A Graph Model of Program 4 (Euclidean Division Algorithm)

Program 4 takes two positive integers $X \in [1, M]$ and $Y \in [1, M]$ as the input and returns their greatest common divisor by implementing the Euclidean Division algorithm. Note that the Main function in Program 4 uses the IntegerDivision program (Program 1).

A. Global Analysis of a Graph Model

A global model can be constructed by embedding the dynamics of the IntegerDivision program within the dynamics of Main. A labeled graph model is shown in Figure 2. This model has a state space $X = \mathcal{N} \times [-M, M]^7$, where $\mathcal{N}$ is the set of nodes as shown in the graph, and the global state $x = [X,\ Y,\ rem,\ dd,\ dr,\ q,\ r]$ is an element of the hypercube $[-M, M]^7$. A *reduced* graph model can be obtained by combining the effects of consecutive transitions and relabeling the reduced graph model accordingly. While analysis of the full graph model is possible, working with a *reduced* model is computationally advantageous. Furthermore, mapping the properties of the reduced graph model to the original model is algorithmic. Interested readers may consult [75] for further elaboration on this topic.

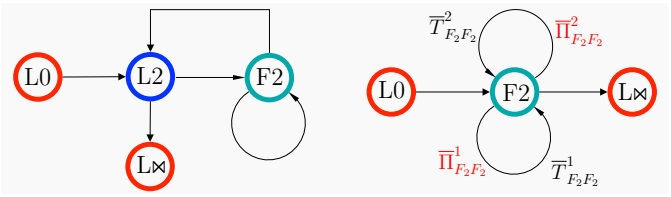

Fig. 3. Two reduced models of the graph model of Program 4.

For the graph model of Program 4, a reduced model can be obtained by first eliminating nodes $F_⋈$, L_4, L_5, L_3, F_0, F_1, F_3, F_4, and L_1, (Figure 3 (Left)) and composing the transition and predicate labels. Node L_2 can then be eliminated as well to obtain a further reduced model with only three nodes: F_2, L_0, and $L_⋈$ (Figure 3 (Right)). This is the model that we analyze. The predicate and transition labels associated with the reduced model are as follows:

$$\overline{T}^1_{F2F2} : x \mapsto [X,\ Y,\ rem,\ dd,\ dr,\ q+1,\ r-dr]$$
$$\overline{T}^2_{F2F2} : x \mapsto [Y,\ r,\ r,\ Y,\ r,\ 0,\ Y]$$
$$\overline{T}_{L0F2} : x \mapsto [X,\ Y,\ 0,\ X,\ Y,\ 0,\ X]$$
$$\overline{T}_{F2L⋈} : x \mapsto [Y,\ r,\ r,\ dd,\ dr,\ q,\ r]$$
$$\Pi^2_{F2F2} : \{x \mid 1 \le r \le dr - 1\}$$
$$\Pi^1_{F2F2} : \{x \mid r \ge dr\}, \qquad \Pi_{F2L⋈} : \{x \mid r \le dr - 1,\ r \le 0\}$$

Finally, the invariant sets that can be readily included in the graph model are (c.f. Appendix I):

$$X_{L0} = \{x \mid M \ge X,\ M \ge Y,\ X \ge 1,\ Y \ge 1\},$$
$$X_{F2} = \{x \mid dd = X,\ dr = Y\}, \quad X_{L⋈} = \{x \mid Y \le 0\}.$$

We are interested in generating certificates of termination and absence of overflow. First, by recursively searching for linear invariants we are able to establish simple lower bounds on all variables in just two rounds (the properties established in each round are added to the model and the search is repeated). For instance, the property $X \ge 1$ is established only after $Y \ge 1$ is established. These results, which were obtained by applying the first part of Theorem 3 (equations (24)-(25) only) with linear functionals, are summarized in Table II.

TABLE II
LINEAR INVARIANTS CORRESPONDING TO PROGRAM 4

Property	$\mathrm{q}\geq 0$	$\mathrm{Y}\geq 1$	$\mathrm{dr}\geq 1$	$\mathrm{rem}\geq 0$	$\mathrm{dd}\geq 1$	$\mathrm{X}\geq 1$	$\mathrm{r}\geq 0$
Proven in Round	I	I	I	I	II	II	II
$\sigma_{\mathrm{F}_2}(\mathrm{x})=$	$-\mathrm{q}$	$1-\mathrm{Y}$	$1-\mathrm{dr}$	$-\mathrm{rem}$	$1-\mathrm{dd}$	$1-\mathrm{X}$	$-\mathrm{r}$
$(\theta^1_{\mathrm{F2F2}},\mu^1_{\mathrm{F2F2}})$	$(1,1)$	$(1,0)$	$(1,0)$	$(1,0)$	$(0,0)$	$(0,0)$	$(0,0)$
$(\theta^2_{\mathrm{F2F2}},\mu^2_{\mathrm{F2F2}})$	$(0,0)$	$(0,0)$	$(0,0)$	$(0,0)$	$(0,0)$	$(0,0)$	$(0,0)$

We then add these properties to the node invariant sets to obtain stronger invariants that certify FTT and boundedness of all variables in $[-\mathrm{M},\mathrm{M}]$. By applying Theorem 3 and SOS programming using YALMIP [88], the following invariants are found[2] (after post-processing, rounding the coefficients, and reverifying):

$$\begin{aligned}
\sigma_{1\mathrm{F}_2}(\mathrm{x}) &= 0.4\,(\mathrm{Y}-\mathrm{M})\,(2+\mathrm{M}-\mathrm{r})\\
\sigma_{2\mathrm{F}_2}(\mathrm{x}) &= (\mathrm{q}\times\mathrm{Y}+\mathrm{r})^2-\mathrm{M}^2\\
\sigma_{3\mathrm{F}_2}(\mathrm{x}) &= (\mathrm{q}+\mathrm{r})^2-\mathrm{M}^2\\
\sigma_{4\mathrm{F}_2}(\mathrm{x}) &= 0.1\left(\mathrm{Y}-\mathrm{M}+5\mathrm{Y}\times\mathrm{M}+\mathrm{Y}^2-6\mathrm{M}^2\right)\\
\sigma_{5\mathrm{F}_2}(\mathrm{x}) &= \mathrm{Y}+\mathrm{r}-2\mathrm{M}+\mathrm{Y}\times\mathrm{M}-\mathrm{M}^2\\
\sigma_{6\mathrm{F}_2}(\mathrm{x}) &= \mathrm{r}\times\mathrm{Y}+\mathrm{Y}-\mathrm{M}^2-\mathrm{M}
\end{aligned}$$

The properties proven by these invariants are summarized in Table III. The two specifications that the program terminates, and that $\mathrm{x}\in[-\mathrm{M},\mathrm{M}]^7$ for all initial conditions X, Y$\in[1,\mathrm{M}]$, could not be established in one shot, at least when trying polynomials of degree $d\leq 4$. For instance, $\sigma_{1\mathrm{F}_2}$ certifies boundedness of all the variables except q, while $\sigma_{2\mathrm{F}_2}$ and $\sigma_{3\mathrm{F}_2}$ which certify boundedness of all variables including q do not certify FTT. Furthermore, boundedness of some of the variables is established in round II, relying on boundedness properties proven in round I. Given $\sigma(\mathrm{x})\leq 0$ (which is found in round I), second round verification can be done by searching for a strictly positive polynomial $p(\mathrm{x})$ and a nonnegative polynomial $q(\mathrm{x})\geq 0$ satisfying:

$$q(\mathrm{x})\,\sigma(\mathrm{x})-p(\mathrm{x})\left(\left(\overline{T}\mathrm{x}\right)_i^2-\mathrm{M}^2\right)\geq 0,\quad \overline{T}\in\{\overline{T}^1_{\mathrm{F2F2}},\overline{T}^2_{\mathrm{F2F2}}\}$$

where the above inequality is further subject to boundedness properties established in round I, as well as the usual predicate conditions and basic invariant set conditions.

In conclusion, $\sigma_{2\mathrm{F}_2}(\mathrm{x})$ or $\sigma_{3\mathrm{F}_2}(\mathrm{x})$ in conjunction with $\sigma_{5\mathrm{F}_2}(\mathrm{x})$ or $\sigma_{6\mathrm{F}_2}(\mathrm{x})$ prove finite-time termination of the algorithm, as well as boundedness of all variables within $[-\mathrm{M},\mathrm{M}]$ for all initial conditions $\mathrm{X},\mathrm{Y}\in[1,\mathrm{M}]$, for any $\mathrm{M}\geq 1$. The provable bound on the number of iterations certified by $\sigma_{5\mathrm{F}_2}(\mathrm{x})$ and $\sigma_{6\mathrm{F}_2}(\mathrm{x})$ is $T_u=2\mathrm{M}^2$ (Corollary 2). If we settle for more conservative specifications, e.g., $\mathrm{x}\in[-k\mathrm{M},k\mathrm{M}]^7$ for all initial conditions $\mathrm{X},\mathrm{Y}\in[1,\mathrm{M}]$ and sufficiently large k, then it is possible to prove the properties in one shot. We show this in the next section.

B. Global Analysis of a MIL-GH Model

For comparison, we also present the MIL-GH model associated with the reduced graph in Figure 3. The corresponding matrices are omitted for brevity, but details of the model along with executable Matlab verification codes can be found in [96]. The verification theorem used in this analysis is an extension of Theorem 2 to analysis of MIL-GHM for specific numerical values of M, though it is certainly possible to perform this modeling and analysis exercise with M as a parameter of the model. The analysis using the MIL-GHM is in general more conservative than SOS optimization over the graph model. This can be attributed to the type of relaxations proposed (similar to those used in Proposition 4) for analysis of MILMs and MIL-GHMs. The benefits are simplified analysis at a typically much lower computational cost. The certificate obtained in this way is a single quadratic function (for each numerical value of M), establishing a bound $\gamma(\mathrm{M})$ satisfying

$$\gamma(\mathrm{M})\geq\left(\mathrm{X}^2+\mathrm{Y}^2+\mathrm{rem}^2+\mathrm{dd}^2+\mathrm{dr}^2+\mathrm{q}^2+\mathrm{r}^2\right)^{1/2}$$

Table IV summarizes the results of this analysis which were performed using both SeDuMi [78] and LMILAB [77] solvers.

C. Modular Analysis

The preceding results were obtained by analysis of a global model which was constructed by embedding the internal dynamics of the program's functions within the global dynamics of the Main function. In contrast, the idea in *modular analysis* is to model software as the interconnection of the program's "building blocks" or "modules", i.e., functions that interact via a set of *global* variables. The dynamics of the functions are then abstracted via Input/Output behavioral models, typically constituting equality and/or inequality constraints relating the input and output variables. In our framework, the invariant sets of the terminal nodes of the modules (e.g., the set $X_{\bowtie}$ associated with the terminal node $\mathrm{F}_{\bowtie}$ in Program 4) provide such I/O model. Thus, richer characterization of the invariant sets of the terminal nodes of the modules are desirable. Correctness of each sub-module must be established separately, while correctness of the entire program is established by verifying the unreachability and termination properties w.r.t. the global variables, as well as verifying that a terminal global state will be reached in finite-time. This way, the program variables that are *private* to each function are abstracted away from the global dynamics. This approach has the potential to greatly simplify the analysis and improve the scalability of the proposed framework as analysis of large size computer programs is undertaken. In this section, we apply the framework to modular analysis of Program 4. Detailed analysis of the advantages in terms of improving scalability, and the limitations in terms of applicability and conservatism of such analyses is an important direction for further research.

The first step is to establish correctness of the INTEGERDIVISION module, for which we obtain:

$$\sigma_{7\mathrm{F2}}(\mathrm{dd},\mathrm{dr},\mathrm{q},\mathrm{r})=(\mathrm{q}+\mathrm{r})^2-\mathrm{M}^2$$

The function $\sigma_{7\mathrm{F2}}$ is a $(1,0)$-invariant proving boundedness of the state variables of INTEGERDIVISION. Subject to boundedness, we obtain the function

$$\sigma_{8\mathrm{F2}}(\mathrm{dd},\mathrm{dr},\mathrm{q},\mathrm{r})=2\mathrm{r}-11\mathrm{q}-6\mathrm{Z}$$

[2]Different choices of polynomial degrees for the invariant functions and the multipliers, as well as different choices for θ,μ and different rounding schemes lead to different invariants. Note that rounding is not essential.

TABLE III

RESULTS FROM ANALYSIS OF A GRAPH MODEL OF PROGRAM 4 (SECTION V-A)

Invariant $\sigma_{\mathrm{F}_2}(\mathrm{x}) =$	$\sigma_{1\mathrm{F}_2}(\mathrm{x})$	$\sigma_{2\mathrm{F}_2}(\mathrm{x}), \sigma_{3\mathrm{F}_2}(\mathrm{x})$	$\sigma_{4\mathrm{F}_2}(\mathrm{x})$	$\sigma_{5\mathrm{F}_2}(\mathrm{x}), \sigma_{6\mathrm{F}_2}(\mathrm{x})$
$\left(\theta^1_{\mathrm{F2F2}}, \mu^1_{\mathrm{F2F2}}\right)$	$(1,0)$	$(1,0)$	$(1,0)$	$(1,1)$
$\left(\theta^2_{\mathrm{F2F2}}, \mu^2_{\mathrm{F2F2}}\right)$	$(1,0.8)$	$(0,0)$	$(1,0.7)$	$(1,1)$
Round I: $\mathrm{x}_i^2 \leq \mathrm{M}^2$ for $\mathrm{x}_i =$	$\mathrm{Y}, \mathrm{X}, \mathrm{r}, \mathrm{dr}, \mathrm{rem}, \mathrm{dd}$	$\mathrm{q}, \mathrm{Y}, \mathrm{dr}, \mathrm{rem}$	$\mathrm{Y}, \mathrm{X}, \mathrm{r}, \mathrm{dr}, \mathrm{rem}, \mathrm{dd}$	$\mathrm{Y}, \mathrm{dr}, \mathrm{rem}$
Round II: $\mathrm{x}_i^2 \leq \mathrm{M}^2$ for $\mathrm{x}_i =$		$\mathrm{X}, \mathrm{r}, \mathrm{dd}$		$\mathrm{X}, \mathrm{r}, \mathrm{dd}$
Certificate for FTT	NO	NO	NO	YES, $T_u = 2\mathrm{M}^2$

TABLE IV

RESULTS FROM ANALYSIS OF A MIL-GHM OF PROGRAM 4 (SECTION V-B)

M	10^2	10^3	10^4	10^5	10^6
Solver: LMILAB [77]: $\gamma(\mathrm{M})$	5.99M	6.34M	6.43M	6.49M	7.05M
Solver: SeDuMi 1_3 [78]: $\gamma(\mathrm{M})$	6.00M	6.34M	6.44M	6.49M	NAN
$\left(\theta^1_{\mathrm{F2F2}}, \mu^1_{\mathrm{F2F2}}\right)$	$\left(1, 10^{-3}\right)$	$\left(1, 10^{-3}\right)$	$\left(1, 10^{-3}\right)$	$\left(1, 10^{-3}\right)$	$\left(1, 10^{-3}\right)$
$\left(\theta^2_{\mathrm{F2F2}}, \mu^2_{\mathrm{F2F2}}\right)$	$\left(1, 10^{-3}\right)$	$\left(1, 10^{-3}\right)$	$\left(1, 10^{-3}\right)$	$\left(1, 10^{-3}\right)$	$\left(1, 10^{-3}\right)$
Upperbound on iterations	$T_u =$ 2e4	$T_u =$ 8e4	$T_u =$ 8e5	$T_u =$ 1.5e7	$T_u =$ 3e9

which is a $(1,1)$-invariant proving termination of INTEGERDIVISION. The invariant set of node $\mathrm{F}_{\bowtie}$ can thus be characterized by

$$X_{\bowtie} = \left\{(\mathrm{dd}, \mathrm{dr}, \mathrm{q}, \mathrm{r}) \in [0, \mathrm{M}]^4 \mid \mathrm{r} \leq \mathrm{dr} - 1\right\}$$

The next step is construction of a global model. Given $X_{\bowtie}$, the transition label at L3: $\mathrm{rem} \leftarrow \mathrm{IntegerDivision}\ (\mathrm{X}\ ,\ \mathrm{Y})$ can be abstracted by

$$\mathrm{rem} = \mathrm{W}, \text{ subject to } \mathrm{W} \in [0, \mathrm{M}],\ \mathrm{W} \leq \mathrm{Y} - 1,$$

allowing for construction of a global model with variables $\mathrm{X}, \mathrm{Y},$ and rem, and an external state-dependent input $\mathrm{W} \in [0, \mathrm{M}]$, satisfying $\mathrm{W} \leq \mathrm{Y} - 1$. The final step is analysis of the global model. We obtain the function $\sigma_{9\mathrm{L}2}(\mathrm{X}, \mathrm{Y}, \mathrm{rem}) = \mathrm{Y} \times \mathrm{M} - \mathrm{M}^2$, which is $(1,1)$-invariant proving both FTT and boundedness of all variables within $[\mathrm{M}, \mathrm{M}]$.

VI. CONCLUDING REMARKS AND FUTURE WORK

We took a systems-theoretic approach to software analysis, and presented a framework based on convex optimization of Lyapunov invariants for verification of a range of important specifications for software systems, including finite-time termination and absence of run-time errors such as overflow, out-of-bounds array indexing, division-by-zero, and user-defined program assertions. The verification problem is reduced to solving a numerical optimization problem which, when feasible, results in a certificate for the desired specification. The novelty of the framework, and consequently, the main contributions of this paper are in the systematic transfer of Lyapunov functions and the associated computational techniques from control systems to software analysis.

The proposed framework provides a constructive method, i.e., a generic algorithm with polynomial time complexity in the dimension of the state space of the input program, to a provably intractable problem. As such, the framework is not guaranteed to succeed, except for programs with relatively simple dynamics (e.g., linear) for which a certain class of Lyapunov functions (e.g., quadratic) are known to exist, and relatively simple constraint structures for which convex relaxation techniques are provably lossless. However, similar to converse Lyapunov theorems in stability analysis of nonlinear dynamical systems [52], converse theorems establishing the necessity of existence of smooth Lyapunov invariants for proving unreachability and termination properties are within reach. One can then hope to find these functions within the class of polynomial or rational functions using polynomial optimization techniques. Some recent progress has been made on establishing existence of finitely-parameterized, low-complexity polynomial Lyapunov functions for exponentially stable systems with smooth polynomial dynamics [97]. Nevertheless, theoretical results in this direction are in general difficult to obtain for systems with more complicated dynamics and for other forms of stability, as desired in software systems. Naturally, our framework inherits the same difficulties, making it even more challenging to prove generic statements about conservativeness of the framework. The attractive feature of the framework is its reliance on both a theory which has demonstrated success in establishing properties of relatively complex systems in diverse application domains, e.g., networks [98], systems biology [99], and engineering [84], using relatively simple behavior certificates, and the recent advances in numerical optimization which strengthen applications of the theory.

The presented work can be extended in several directions. These include understanding the power and the limitations of modular analysis of programs, robustness analysis of the (nominal) Lyapunov certificates to model perturbations induced by round-off errors, extension to systems with software in closed loop with hardware, and adaptation of the framework to specific classes of software in specific applications.

APPENDIX I
CONSTRUCTION OF SIMPLE INVARIANT SETS

Simple invariant sets can be included in the model if they are readily available or easily computable. Even trivial invariants

can simplify the analysis and improve the chances of finding stronger invariants via convex relaxations. Before we proceed, we introduce the following notation: Given a semialgebraic set Π, and a polynomial function $\tau : \mathbb{R}^n \mapsto \mathbb{R}^n$, we denote by $\Pi(\tau)$, the set: $\Pi(\tau) = \{x \mid \tau(x) \in \Pi\}$.

- Simple invariant sets may be provided by the programmer. These can be trivial sets representing simple algebraic relations between variables, or they can be complicated relations that reflect the programmer's knowledge about functionality and behavior of the program.
- Invariant Propagation: Assuming that T_{ij}^k are deterministic and invertible, the set

$$X_i = \bigcup_{j\in\mathcal{I}(i),\ k\in\mathcal{A}_{ij}} \Pi_{ij}^k\left(\left(T_{ij}^k\right)^{-1}\right) \tag{27}$$

is an invariant set for node i. Furthermore, if the invariant sets X_j are strict subsets of Ω^n for all $j \in \mathcal{I}(i)$, then (27) can be improved. Specifically, the set

$$X_i = \bigcup_{j\in\mathcal{I}(i),\ k\in\mathcal{A}_{ij}} \Pi_{ij}^k\left(\left(T_{ij}^k\right)^{-1}\right) \cap X_j\left(\left(T_{ij}^k\right)^{-1}\right)$$

is an invariant set for node i. Note that it is sufficient that the restriction of T_{ij}^k to the lower dimensional spaces in the domains of Π_{ij}^k and X_j be invertible.
- Preserving Equality Constraints: Simple assignments of the form $T_{ij}^k : x_l \mapsto f(x_m)$ result in invariant sets at node i, of the form $X_i = \{x \mid x_l - f(x_m) = 0\}$, provided that T_{ij}^k does not simultaneously update x_m. Formally, let T_{ij}^k be such that $(T_{ij}^k x)_l - x_l$ is non-zero for at most one element $\hat{l} \in \mathbb{Z}(1,n)$, and that $(T_{ij}^k x)_{\hat{l}}$ is independent of $x_{\hat{l}}$. Then, the following set:

$$X_i = \bigcup_{j\in\mathcal{I}(i),\ k\in\mathcal{A}_{ij}} \left\{x \mid \left[T_{ij}^k - I\right]x = 0\right\},$$

is an invariant set at node i.

APPENDIX II

Proof of Theorem 1: Assume that $\mathcal{S}$ has a solution $\mathcal{X}=(x(0),...,x(t_-),...)$, where $x(0) \in X_0$ and $x(t_-) \in X_-$. Let

$$\gamma_h = \inf_{x\in h^{-1}(X_-)} V(x)$$

First, we claim that $\gamma_h \le \max\{V(x(t_-)), V(x(t_- - 1))\}$. If $h = I$, we have $x(t_-) \in h^{-1}(X_-)$ and $\gamma_h \le V(x(t_-))$. If $h = f$, we have $x(t_- - 1) \in h^{-1}(X_-)$ and $\gamma_h \le V(x(t_- - 1))$, hence the claim. Now, consider the $\theta = 1$ case. Since V is monotonically decreasing along solutions of $\mathcal{S}$, we must have:

$$\begin{aligned}\gamma_h = \inf_{x\in h^{-1}(X_-)} V(x) &\le \max\{V(x(t_-)), V(x(t_- - 1))\}\\ &\le V(x(0)) \le \sup_{x\in X_0} V(x)\end{aligned} \tag{28}$$

which contradicts (11). Note that if $\mu > 0$ and $h = I$, then (28) holds as a strict inequality and we can replace (11) with its non-strict version. Next, consider case (I), for which, V need not be monotonic along the trajectories. Partition X_0 into two subsets $\overline{X}_0$ and $\underline{X}_0$ such that $X_0 = \overline{X}_0 \cup \underline{X}_0$ and

$$V(x) \le 0 \quad \forall x \in \underline{X}_0, \qquad \text{and} \qquad V(x) > 0 \quad \forall x \in \overline{X}_0$$

Now, assume that $\mathcal{S}$ has a solution $\overline{\mathcal{X}}=(\overline{x}(0),...,\overline{x}(t_-),...)$, where $\overline{x}(0) \in \overline{X}_0$ and $\overline{x}(t_-) \in X_-$. Since $V(x(0)) > 0$ and $\theta < 1$, we have $V(x(t)) < V(x(0)), \quad \forall t > 0$. Therefore,

$$\begin{aligned}\gamma_h = \inf_{x\in h^{-1}(X_-)} V(x) &\le \max\{V(x(t_-)), V(x(t_- - 1))\}\\ &\le V(\overline{x}(0)) \le \sup_{x\in X_0} V(x)\end{aligned}$$

which contradicts (11). Next, assume that $\mathcal{S}$ has a solution $\underline{\mathcal{X}}=(\underline{x}(0),...,\underline{x}(t_-),...)$, where $\underline{x}(0) \in \underline{X}_0$ and $\underline{x}(t_-) \in X_-$. In this case, regardless of the value of θ, we must have $V(\underline{x}(t)) \le 0$, $\forall t$, implying that $\gamma_h \le 0$, and hence, contradicting (12). Note that if $h = I$ and either $\mu > 0$, or $\theta > 0$, then (12) can be replaced with its non-strict version. Finally, consider case (II). Due to (13), V is strictly monotonically decreasing along the solutions of $\mathcal{S}$. The rest of the argument is similar to the $\theta = 1$ case. ∎

Proof of Corollary 1: It follows from (15) and the definition of X_- that:

$$\begin{aligned}V(x) &\ge \sup\left\{\left\|\alpha^{-1}h(x)\right\|_q - 1\right\}\\ &\ge \sup\left\{\left\|\alpha^{-1}h(x)\right\|_\infty - 1\right\} > 0, \qquad \forall x \in X.\end{aligned} \tag{29}$$

It then follows from (29) and (14) that:

$$\inf_{x\in h^{-1}(X_-)} V(x) > 0 \ge \sup_{x\in X_0} V(x)$$

Hence, the first statement of the Corollary follows from Theorem 1. The upperbound on the number of iterations follows from Proposition 3 and the fact that $\sup_{x\in X\setminus\{X_-\cup X_\infty\}} |V(x)| \le 1$. ∎

Proof of Corollary 2: The unreachability property follows directly from Theorem 1. The finite time termination property holds because it follows from (6), (17) and (20c) along with Proposition 3, that the maximum number of iterations around every simple cycle $\mathcal{C}$ is finite. The upperbound on the number of iterations is the sum of the maximum number of iterations over every simple cycle. ∎

Proof of Proposition 4: Define $x_e = (x, w, v, 1)^T$, where $x \in [-1,1]^n$, $w \in [-1,1]^{n_w}$, $v \in \{-1,1\}^{n_v}$. Recall that $(x,1)^T = L_2 x_e$, and that for all x_e satisfying $Hx_e = 0$, there holds: $(x_+, 1) = (Fx_e, 1) = L_1 x_e$. It follows from Proposition 2 that (4) holds if:

$$\begin{aligned}&x_e^T L_1^T P L_1 x_e - \theta x_e^T L_2^T P L_2 x_e \le -\mu, \text{ for all } x_e \text{ such that:}\\ &Hx_e = 0,\ L_3 x_e \in [-1,1]^{n+n_w},\ L_4 x_e \in \{-1,1\}^{n_v}.\end{aligned} \tag{30}$$

Recall from the $\mathcal{S}$-Procedure (Sec. IV-A2a) that the assertion $\sigma(y) \le 0$, $\forall y \in [-1,1]^n$ holds if there exist nonnegative constants $\tau_i \ge 0$, $i = 1,...,n$, such that $\sigma(y) \le \sum \tau_i\left(y_i^2 - 1\right) = y^T\tau y - \text{Trace}(\tau)$, where $\tau = \text{diag}\{\tau_i\} \succeq 0$. Similarly, the assertion $\sigma(y) \le 0, \forall y \in \{-1,1\}^n$ holds if there exist constants ρ_i such that $\sigma(y) \le \sum \rho_i\left(y_i^2 - 1\right) = y^T\rho y - \text{Trace}(\rho)$, where $\rho = \text{diag}\{\rho_i\}$. Applying these relaxations to (30), we obtain sufficient conditions for (30) to hold:

$$\begin{aligned}&x_e^T L_1^T P L_1 x_e - \theta x_e^T L_2^T P L_2 x_e \le x_e^T\left(YH + H^T Y^T\right)x_e +\\ &x_e^T L_3^T D_{xw} L_3 x_e + x_e^T L_4^T D_v L_4 x_e - \mu - \text{Trace}(D_{xw} + D_v)\end{aligned}$$

Together with $D_{xw} \succeq 0$, the above condition is equivalent to the LMIs in Proposition 4. ∎

REFERENCES

[1] H. Kopetz, *Real-Time Systems Design Principles for Distributed Embedded Applications*. Kluwer, 2001.

[2] C. S. Murthy and G. Manimaran, *Resource Management in Real-Time Systems and Networks*. MIT Press, 2001.

[3] B. Heck, L. Wills, and G. Vachtsevanos, "Software technology for implementing reusable, distributed control systems," *Control Syst. Mag., IEEE*, vol. 23, pp. 21–35, Feb. 2003.

[4] D. A. Peled, *Software Reliability Methods*. Springer, 2001.

[5] F. Nielson, H. Nielson, and C. Hank, *Principles of Program Analysis*. Springer, 2004.

[6] J. F. Monin, *Understanding Formal Methods*. Springer, 2003.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 1999.

[8] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 3rd ed., 2006.

[9] D. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, *The Theory of Timed I/O Automata*. Morgan and Claypool, 2nd ed., 2010.

[10] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[11] Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

[12] E. M. Clarke *et al.*, "Verification of the futurebus+cache coherence protocol," *Formal Methods in System Design*, vol. 6, no. 2, pp. 217–232, 1995.

[13] W. Marrero, E. Clarke, and S. Jha, "Model checking for security protocols," Tech. Rep. CMU-SCS-97-139, Carnegie Mellon Univ., 1997.

[14] G. Naumovich, L. A. Clarke, and L. J. Osterweil, "Verification of communication protocols using data flow analysis," in *Proc. the 4th ACM Symp. on the Foundation of Software Eng.*, pp. 93–105, 1996.

[15] C. Baier *et al.*, "Model-checking algorithms for continuous-time Markov chains," *Software Eng., IEEE Trans. on*, vol. 29, pp. 524–541, Jun. 2003.

[16] R. Bryant, "Graph-based algorithms for boolean function manipulation," *Comput., IEEE Trans. on*, vol. C-35, pp. 677–691, Aug. 1986.

[17] M. W. Moskewicz *et al.*, "Chaff: engineering an efficient SAT solver," in *Proc. of the 38th Annu. Design Automation Conf.*, DAC '01, pp. 530–535, 2001.

[18] L. de Moura and N. Bjrner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Anal. of Syst.*, vol. 4963 of *LNCS*, pp. 337–340, Springer, 2008.

[19] R. Alur, T. Dang, and F. Ivančić, "Predicate abstraction for reachability analysis of hybrid systems," *ACM Trans. on Embedded Computing Systems*, vol. 5, pp. 152–199, Feb. 2006.

[20] D. Dams, *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Eindhoven Univ. of Tech., 1996.

[21] A. Tiwari and G. Khanna, "Series of abstractions for hybrid automata," in *Hybrid Syst.: Computation and Control*, vol. 2289 of *LNCS*, pp. 465–478, Springer, 2002.

[22] S. Sankaranarayanan and A. Tiwari, "Relational abstractions for continuous and hybrid systems," in *Proc. of the 23rd Int. Conf. on Comput. aided verification*, CAV'11, pp. 686–702, 2011.

[23] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Proc. of the 4th ACM symp. on Principles of Programming Languages*, POPL '77, pp. 238–252, ACM, 1977.

[24] P. Cousot, "Abstract interpretation based formal methods and future challenges," in *Informatics - 10 Years Back. 10 Years Ahead.*, pp. 138–156, Springer-Verlag, 2001.

[25] A. Adje, *Optimisation et jeux appliqués à l'analyse statique de programmes par interprétation abstraite*. PhD thesis, Ecole Polytechnique, France, 2011.

[26] URL= http://www.astree.ens.fr/ , retrieved 03/26/2012.

[27] J. Bertrane *et al.*, "Static analysis and verification of aerospace software by abstract interpretation," in *AIAA Infotech@Aerospace (I@A 2010)*, pp. 1–38, Apr. 2010.

[28] E. Goubault, "Static analyses of the precision of floating-point operations," in *Static Anal.*, vol. 2126 of *LNCS*, pp. 234–259, Springer, 2001.

[29] A. Venet and G. Brat, "Precise and efficient static array bound checking for large embedded C programs," in *Proc. of the ACM SIGPLAN 2004 Conf. on Programming Language Design and Implementation*, PLDI '04, pp. 231–242, ACM, 2004.

[30] A. Costan *et al.*, "A policy iteration algorithm for computing fixed points in static analysis of programs," in *Comput. Aided Verification*, vol. 3576 of *LNCS*, pp. 215–226, Springer, 2005.

[31] J. Feret, "Static analysis of digital filters," in *European Symp. on Programming (ESOP'04)*, no. 2986 in LNCS, Springer-Verlag, 2004.

[32] B. C. Pierce, *Types and Programming Languages*. MIT Press, 2002.

[33] M. Hecht, *Flow Analysis of Computer Programs*. Elsevier Science, 1977.

[34] S. Gulwani, S. Srivastava, and R. Venkatesan, "Program analysis as constraint solving," in *Proc. of the 2008 ACM Conf. on Programming Language Design and Implementation*, PLDI '08, pp. 281–292, 2008.

[35] T. Ball, V. Levin, and S. K. Rajamani, "A decade of software model checking with SLAM," *Commun. of the ACM*, vol. 54, no. 7, pp. 68–76, 2011.

[36] T. Kailath, *Linear Systems*. Prentice-Hall, 1980.

[37] A. M. Lyapunov, *Problème général de la stabilité du mouvement*, vol. 17 of *Ann. of Math. Studies*. Princeton University Press, 1947.

[38] V. A. Yakubovich, "The solution of certain matrix inequalities in automatic control theory," *Soviet Math. Dokl.*, vol. 3, pp. 620–623, 1962. In Russian, 1961.

[39] J. Doyle, "Analysis of feedback systems with structured uncertainties," *Control Theory and Applicat., IEE Proc. D*, vol. 129, pp. 242–250, Nov. 1982.

[40] G. Pappas, G. Lafferriere, and S. Sastry, "Hierarchically consistent control systems," *IEEE Trans. Autom. Control*, vol. 45, pp. 1144–1160, Jun. 2000.

[41] E. Haghverdi, P. Tabuada, and G. Pappas, "Bisimulation relations for dynamical, control, and hybrid systems," *Theoretical Comput. Sci.*, vol. 342, pp. 229–261, 2005.

[42] A. Girard, G. Pola, and P. Tabuada, "Approximately bisimilar symbolic models for incrementally stable switched systems," *IEEE Trans. Autom. Control*, vol. 55, no. 1, pp. 116–126, 2010.

[43] A. Girard and G. J. Pappas, "Approximate bisimulations for constrained linear systems," in *4th IEEE Conf. on Decision and Control, and 2005 European Control Conf. CDC-ECC '05*, pp. 4700– 4705, Dec. 2005.

[44] G. Lafferriere, G. J. Pappas, and S. Yovine, "Symbolic reachability computation for families of linear vector fields," *J. of Symbolic Computation.*, vol. 32, no. 3, pp. 231–253, 2001.

[45] A. B. Kurzhanski and I. Valyi, *Ellipsoidal Calculus for Estimation and Control*. Birkhauser, 1996.

[46] A. Bemporad, F. D. Torrisi, and M. Morari, "Optimization-based verification and stability characterization of piecewise affine and hybrid systems," in *Int. Workshop on Hybrid Syst.: Computation and Control*, vol. 1790 of *LNCS*, pp. 45–58, Springer-Verlag, Mar. 2000.

[47] S. Prajna, A. Jadbabaie, and G. Pappas, "A framework for worst-case and stochastic safety verification using barrier certificates," *IEEE Trans. Autom. Control*, vol. 52, pp. 1415–1428, Aug. 2007.

[48] M. Branicky, "Multiple Lyapunov functions and other analysis tools for switched and hybrid systems," *IEEE Trans. Autom. Control*, vol. 43, pp. 475–482, Apr. 1998.

[49] M. Johansson and A. Rantzer, "Computation of piecewise quadratic Lyapunov functions for hybrid systems," *IEEE Trans. Autom. Control*, vol. 43, pp. 555 –559, Apr. 1998.

[50] A. Girard and G. Pappas, "Verification using simulation," in *HSCC 2006*, vol. 3927 of *LNCS*, pp. 272–286, Springer, 2006.

[51] G. Lafferriere, G. J. Pappas, and S. Sastry, "Hybrid systems with finite bisimulations," in *Hybrid Syst. V*, vol. 1567 of *LNCS*, pp. 186–203, Springer-Verlag, 1999.

[52] K. H. Khalil, *Nonlinear Systems*. Prentice Hall, 2002.

[53] S. Prajna and A. Rantzer, "Convex programs for temporal verification of nonlinear dynamical systems," *SIAM J. on Control and Optimization*, vol. 46, no. 3, pp. 999–1021, 2007.

[54] E. Feron, "Abstraction mechanisms across the board: A short introduction," Mar. 18, 2004. Presentations available at: http://web.mit.edu/feron/Public/wkshop_pres , retrieved 03/26/2012.

[55] M. Roozbehani, A. Megretski, and E. Feron, "Convex optimization proves software correctness," in *American Control Conf., 2005. Proceedings of the 2005*, pp. 1395 – 1400, Jun. 2005.

[56] M. Roozbehani, E. Feron, and A. Megretski, "Modeling, optimization and computation for software verification," in *Hybrid Syst.: Computation and Control*, vol. 3414 of *LNCS*, pp. 606–622, Springer, 2005.

[57] M. Roozbehani, A. Megretski, and E. Feron, "Safety verification of iterative algorithms over polynomial vector fields," in *Decision and Control, 2006, Proc. 45th IEEE Conf. on*, pp. 6061–6067, 2006.

[58] P. Cousot, "Proving program invariance and termination by parametric abstraction, Lagrangian relaxation and semidefinite programming," in *Proc. 6th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, VMCAI'05, pp. 1–24, Springer-Verlag, 2005.

[59] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization*. Athena Scientific, 1st ed., 1997.

[60] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM Review*, vol. 38, no. 1, pp. 49–95, 1996.

[61] J. B. Lasserre, *Moments, Positive Polynomials and their Applications*. Imperial College Press, 2009.
[62] M. Laurent, "Sums of squares, moment matrices and optimization over polynomials," in *Emerging Applications of Algebraic Geometry*, vol. 149 of *IMA Volumes in Math. and its Applicat.*, pp. 157–270, Springer, 2009.
[63] P. A. Parrilo, "Semidefinite programming relaxations for semialgebraic problems," *Math. Programming*, vol. 96, no. 2, pp. 293–320, 2003.
[64] M. Roozbehani, A. Megretski, and E. Feron, "Optimization of Lyapunov invariants in analysis of software systems," tech. rep., MIT, 2011. URL: http://arxiv.org/abs/1108.5622.
[65] A. Miné, "Relational abstract domains for the detection of floating-point run-time errors," in *European symp. on programming (ESOP)*, vol. 2986 of *LNCS*, pp. 3–17, 2004.
[66] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.
[67] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, 1999.
[68] R. Alur *et al.*, "The algorithmic analysis of hybrid systems," *Theoretical Comput. Sci.*, vol. 138, no. 1, pp. 3–34, 1995.
[69] R. Brockett, "Hybrid models for motion control systems," in *Essays on Control: Perspectives in the Theory and its Applicat.*, Birkhauser, 1993.
[70] O. Grumberg and D. E. Long, "Model checking and modular verification," *ACM Trans. on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 3, pp. 843–871, 1994.
[71] B. Cook, A. Podelski, and A. Rybalchenko, "Termination proofs for systems code," in *Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '06, pp. 415–426, 2006.
[72] A. Podelski and A. Rybalchenko, "A complete method for the synthesis of linear ranking functions," in *Verification, Model Checking, and Abstract Interpretation*, VMCAI04, pp. 239–251, Springer, 2004.
[73] A. Megretski and A. Rantzer, "System analysis via integral quadratic constraints," *IEEE Trans. Autom. Control*, vol. 42, pp. 819–830, Jun. 1997.
[74] S. Gulwani, S. Jain, and E. Koskinen, "Control-flow refinement and progress invariants for bound analysis," in *Programming Language Design and Implementation*, PLDI '09, pp. 375–385, 2009.
[75] M. Roozbehani *et al.*, "Distributed Lyapunov functions in analysis of graph models of software," in *Hybrid Syst.: Computation and Control*, vol. 4981 of *LNCS*, pp. 443–456, Springer, 2008.
[76] A. A. Ahmadi and P. A. Parrilo, "Non-monotonic Lyapunov functions for stability of discrete time nonlinear and switched systems," in *Decision and Control, 2008. Proc. 47th IEEE Conf. on*, pp. 614–621, Dec. 2008.
[77] P. Gahinet, A. Nemirovskii, and A. Laub, "LMILAB: A package for manipulating and solving LMIs," 1994.
[78] J. F. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization Methods and Software*, vol. 11, no. 12, pp. 625–653, 1999. URL: http://sedumi.ie.lehigh.edu/.
[79] "IBM ILOG CPLEX Optimization Studio."
[80] L. Lovasz and A. Schrijver, "Cones of matrices and set-functions and 0-1 optimization," *SIAM J. on Optim.*, vol. 1, no. 2, pp. 166–190, 1991.
[81] A. Megretski, "Relaxations of quadratic programs in operator theory and system analysis," *Operator Theory: Advances and Applicat.*, vol. 129, pp. 365–392, 2001.
[82] Y. E. Nesterov, H. Wolkowicz, and Y. Ye, "Semidefinite programming relaxations of nonconvex quadratic optimization," in *Handbook of Semidefinite Programming: Theory, Algorithms, and Applicat.*, vol. 27 of *Int. Series in Operations Research & Manage. Sci.*, pp. 361–419, Springer, 2000.
[83] V. A. Yakubovich, "S-procedure in nonlinear control theory," *Vestnik Leningrad Univ.*, vol. 4, no. 1, pp. 73–93, 1977.
[84] P. A. Parrilo, *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Inst. of Tech., 2000.
[85] S. V. Gusev and A. L. Likhtarnikov, "Kalman–Popov–Yakubovich lemma and the $\mathcal{S}$-procedure: A historical essay," *Automation and Remote Control*, vol. 67, no. 11, pp. 1768–1810, 2006.
[86] J. Bochnak, M. Coste, and M. F. Roy, *Real Algebraic Geometry*. Springer, 1998.
[87] S. Prajna *et al.*, "SOSTOOLS: Sum of squares optimization toolbox for MATLAB," 2004. URL: http://www.mit.edu/~parrilo/sostools.
[88] J. Löfberg, "YALMIP : A toolbox for modeling and optimization in MATLAB," in *Proc. of the CACSD Conf.*, 2004. http://users.isy.liu.se/johanl/yalmip , retrieved 03/26/2012.
[89] H. Waki *et al.*, "Algorithm 883: Sparsepop—a sparse semidefinite programming relaxation of polynomial optimization problems," *ACM Trans. Math. Softw.*, vol. 35, no. 2, 2008.
[90] D. Henrion and J.-B. Lasserre, "GloptiPoly: global optimization over polynomials with MATLAB and SeDuMi," in *Decision and Control, 2002, Proc. of the 41st IEEE Conf. on*, pp. 747–752, Dec. 2002.
[91] A. Megretski, "Positivity of trigonometric polynomials," in *Decision and Control, 2003. Proc. 42nd IEEE Conf. on*, pp. 3814–3817, Dec. 2003.
[92] D. Henrion and A. Garulli, eds., *Positive Polynomials in Control*, vol. 312 of *Lecture Notes in Comput. and Inform. Sci.* Springer, 2005.
[93] G. Chesi and D. Henrion, "Guest editorial: special issue on positive polynomials in control," *IEEE Trans. Autom. Control*, vol. 54, pp. 935–936, May 2009.
[94] M. X. Goemans and D. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *J. of the ACM*, vol. 42, no. 6, pp. 1115–1145, 1995.
[95] H. D. Sherali and W. P. Adams, "A hierarchy of relaxations and convex hull characterizations for mixed-integer zero-one programming problems," *Discrete Applied Math.*, vol. 52, no. 1, pp. 83–106, 1994.
[96] http://web.mit.edu/mardavij/www/Software , retrieved 03/26/2012.
[97] M. M. Peet and A. Papachristodoulou, "A converse sum of squares Lyapunov result with a degree bound," *IEEE Trans. Autom. Control,*, vol. 57, pp. 2281–2293, Sep. 2012.
[98] I. Lestas and G. Vinnicombe, "Scalable decentralized robust stability certificates for networks of interconnected heterogeneous dynamical systems," *IEEE Trans. Autom. Control*, vol. 51, pp. 1613–1625, Oct. 2006.
[99] M. Jovanovic, M. Arcak, and E. Sontag, "A passivity-based approach to stability of spatially distributed systems with a cyclic interconnection structure," *IEEE Trans. Autom. Control*, vol. 53, pp. 75 –86, Jan. 2008.

Mardavij Roozbehani Received the B.Sc. degree from Sharif University of Technology, Tehran, Iran, in 2000, the M.Sc. degree in Mechanical and Aerospace Engineering from the University of Virginia, Charlottesville, VA in 2003, and the Ph.D. degree in Aeronautics and Astronautics from the Massachusetts Institute of Technology (MIT), Cambridge, MA in 2008. Since 2012, he has held a Principal Research Scientist position at the Laboratory for Information and Decision Systems (LIDS) at MIT, where he previously held postdoctoral, course instructor, and research scientist positions between 2008 and 2011. His main research interests include distributed and networked control systems, software and finite-state systems, and dynamics and economics of power systems with an emphasis on robustness and risk. Dr. Roozbehani is a recipient of the 2007 AIAA graduate award for safety verification of real-time software systems.

Alexandre Megretski (M'94–SM'08) received the Ph.D. degree in control theory in from Leningrad University, Leningrad, Russia, in 1988. He was a Researcher with the Royal Institute of Technology, Stockholm, Sweden, University of Newcastle, Australia, and a faculty member at Iowa State University, Ames. He is now a Professor of electrical engineering at MIT, Cambridge, MA. His research involves nonlinear dynamical systems (identification, analysis, design), validation of hybrid control algorithms, optimization, applications to analog circuits, control of animated objects, and relay systems.

Eric Feron (M'94) studied at Ecole Polytechnique, France, and received the M.S. degree in computer science from Ecole Polytechnique and Ecole Normale Supérieure, France, and the Ph.D. degree in aeronautics and astronautics from Stanford University, Stanford, CA, in 1990 and 1994, respectively. Following an engineering appointment with the ministry of defense, France, he was on the faculty with the Department of Aeronautics and Astronautics, the Massachusetts Institute of Technology, Cambridge, MA, from 1993 to 2005. He is the Dutton-Ducoffe Professor of Aerospace Software Engineering at the Georgia Institute of Technology, Atlanta. His research interests are the application of computer science, control and optimization theories to important aerospace problems, including flight control systems and air transportation. He is a co-author of the monograph Linear Matrix Inequalities in System and Control Theory (SIAM: Philadelphia, PA, 1994), and the English translation of Bézout's General Theory of Algebraic Equations (Princeton Univ.: Princeton, NJ, 2006).