2005 American Control Conference
June 8-10, 2005. Portland, OR, USA

WeC08.1

Convex Optimization Proves Software Correctness

Mardavij Roozbehani, Alexandre Megretski, and Eric Feron

Abstract— **This paper concerns analysis of real-time, safety-critical, embedded software. Software analysis is expected to verify whether the computer code will execute safely, free of run-time errors. The main properties to be analyzed to prove or disprove safe execution include boundedness of all variables and termination of the program in finite-time. Herein the concepts of Lyapunov invariance and associated computational procedures are brought within the context of software analysis. Dynamical system representations of software systems along with specific models that are suitable for analysis via Lyapunov-like functions are developed. General forms for the Lyapunov-like invariants are then constructed in a way to certify the desired properties. Convex optimization methods such as linear programming and/or semidefinite programming are then employed for finding appropriate functions that fit into these general forms and therefore, automatically establish the key properties of software.**

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

I. INTRODUCTION

Software and software enabled control systems are instrumental in the design and implementation of real-time, embedded control systems operating in uncertain environments. Examples include human operated avionic vehicles, autonomous aerospace systems and multiple coordinating UAVs. As functionality and performance of these mission-critical systems rely heavily on software, it is crucially important to verify reliability and correctness of the embedded software. The very least to require is that the software must be free of run-time errors. On the other hand, the dramatically growing complexity of these modern control systems demands equal growth in the complexity of the underlying software. The complexity issue brings significant computational and mathematical challenges in analysis of the embedded software, as far as verification of certain properties is concerned.

Mardavij Roozbehani is currently with the department of Aeronautics and Astronautics at Massachusetts Institute of Technology, Cambridge, MA. E-mail: `mardavij@mit.edu`
Alexandre Megretski is currently an associate professor of Electrical engineering at MIT. E-mail: `ameg@mit.edu`
Eric Feron is currently an associate professor of Aeronautics and Astronautics at MIT. E-mail: `feron@mit.edu`.

The software properties that are critical for safe execution include: (1) absence of variable overflow, (2) absence of 'array index out-of-bounds' calls, (3) termination of the functions and subfunctions and if required, the program itself in finite time. Some additional properties that might be desired in a reliable, safety-critical software include: (4) robustness to uncertain inputs, including feedback from analog systems, (5) validity of certain inequalities relating inputs and outputs, and (6) absence of 'dead-code'.

The first property is in some sense equivalent to the stability property of (nonlinear) dynamical systems, which is known to be undecidable even for the simplest cases (for instance piecewise linear systems). Moreover, it is well known that a general procedure that takes a computer program as an input and correctly decides if the program terminates in finite time does not exist. We therefore, are not aiming at finding computationally efficient algorithms that are guaranteed to work on all instances of computer programs. Instead, we search for efficient algorithms that work reasonably well for most practical instances.

Pioneered by the works of Cousot, perhaps the most noteworthy results in the literature that deal with software verification are based on the notion of *abstract interpretation*. See for instance [4],[6]. See also [7],[8]. Abstract interpretation is a theory for approximating the semantics of software and is used for statically analyzing the dynamical properties of computer programs. According to [4], abstract interpretation is defined as an approximate program semantics derived from the domain of concrete semantic operations by replacing it with a domain of abstract semantic operations. In essence, the abstraction-based techniques perform approximate/abstract symbolic executions of the program, until an inductive assertion which remains invariant under further executions of the program, and thus, a system invariant is found. In order to guarantee convergence, however, these methods employ acceleration/extrapolation techniques such as narrowing (outer approximation) and widening (inner approximation). The introduction of these narrowing or widening operators in usually the main source of conservatism in abstract interpretation-based methods [3].

While analysis of semantic properties of software is under rapid growing attention in the computer science literature, little has appeared on this subject in control engineering literature. In this paper we introduce a systems theoretic framework for establishing correctness (in the sense that some or all of the above properties hold) of software systems. The analysis relies on a dynamical systems interpretation of computer programs, and the main method of verification is based on optimization search for system invariants. Convex relaxations of combinatorial optimization problems, as well as the standard tools of convex optimization, serve as the numerical engines. The novelty of this approach is in the transfer of fundamental concepts (Lyapunov invariants) and associated computational techniques from the control systems analysis arena to software engineering. However, it is possible that these techniques complement existing techniques described earlier within the context of abstract interpretation of programs. Existence or nonexistence of a complementary relation between the two methods is an open question.

The main contribution of this paper is the introduction of a novel approach for automatic verification of computer programs of piece-wise linear semantics. These include single flow programs and gain scheduled piecewise linear systems, used to control physical devices such as aerospace systems or automotive control systems. We present modeling techniques through the introduction of linear-like models that may represent a broad range of computer programs of interest to this paper. We will then introduce and design specific Lyapunov invariants, whose properties guarantee variable bounded-ness as well as other desired properties, such as guaranteed program termination. These Lyapunov invariants are instrumental for the approach taken here. That is, they provide the *behavior certificates* of the computer program relative to the (possibly many) properties to be verified. Finally, we will show how this problem may be formulated as a convex optimization problem, such as a linear program, or possibly a program involving semidefinite constraints.

II. BASIC PRINCIPLES OF AUTOMATED SOFTWARE ANALYSIS

This section introduces the fundamentals of our approach towards verification of computer codes through dynamical system models. Considering computer programs as dynamical systems, we introduce Lyapunov-like functions as *certificates* for the behavior of these systems. We then describe the conditions under which finding appropriate Lyapunov functions may be formulated as a convex optimization problem.

A. Dynamical systems representation of computer programs

A computer program can be viewed as a rule for iterative modification of operating memory, possibly in response to real-time inputs. While computer code for real-time applications is expected to run continuously, some programs are expected to terminate in finite time. In both situations, a computer program can be modeled as a dynamical system.

More specifically, we will consider models defined in general by a *state space* set X with selected subsets $X_0 \subset X$ of *initial states* and $X_\infty \subset X$ of *terminal states*, and by a set-valued function $f: \; X \mapsto 2^X$, such that $f(x) \subset X_\infty$, $\forall x \in X_\infty$. The dynamical system $\mathcal{S} = \mathcal{S}(X, f, X_0, X_\infty)$ defined by X, f, X_0, X_∞ is understood as the set of all sequences $\mathcal{X} = (x(0), x(1), \dots, x(t), \dots)$ of elements from X satisfying

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

Note that the uncertainty in the definition of $x(0)$ represents different dependence on parameters, and the uncertainty in the definition of $x(t+1)$ represents program's ability to respond to real-time inputs. From a dynamical systems viewpoint, analysis of software means verification of certain properties of system (1).

Definition 1: A computer program represented by a dynamical system $\mathcal{S} = \mathcal{S}(X, f, X_0, X_\infty)$ is said to terminate in finite time if every solution

$\mathcal{X} = x(t)$ of (1) satisfies $x(t) \in X_\infty$ for some $t \in \mathbb{Z}_+$. In addition, we say that the state variables remain bounded (do not overflow) if $x(t)$ does not belong to a certain (unsafe) subset X_- of X for every solution $\mathcal{X} = x(t)$ of (1).

B. Abstracted models of computer programs

In addition to *exact* models of software systems, we also introduce *abstracted* models. We say that model $\mathcal{S}(\hat{X}, \hat{f}, \hat{X}_0, \hat{X}_\infty)$ is an *abstraction* of $\mathcal{S}(X, f, X_0, X_\infty)$ if $X \subset \hat{X}$, $X_0 \subset \hat{X}_0$, $X_\infty \subset \hat{X}_\infty$, and $f(x) \subset \hat{f}(x)$ for all $x \in X$. When $\mathcal{S}(X, f, X_0, X_\infty)$ is an exact model of a computer code, the set X is finite (since it represents computer's memory), however, in the abstracted models of computer programs, the state space is not constrained to be a finite set. Abstracted representations are useful for producing simplified models of computer code. Validity of certain properties (such as finite time termination and avoidance of overflow) in an abstraction $\mathcal{S}(\hat{X}, \hat{f}, \hat{X}_0, \hat{X}_\infty)$ implies their validity for the original model $\mathcal{S}(X, f, X_0, X_\infty)$.

When performing calculations with real numbers, a processor represents them in an approximate binary form, which complicates definitions of even simple operations such as addition and scaling. An abstracted model of a program which deals with non-integer arithmetic can be defined in terms of the "true" real numbers, which has the potential to simplify the analysis dramatically.

C. Lyapunov functions as behavior certificates

Definition 2: A Lyapunov function for system (1) is defined to be a function $V: X \mapsto \mathbb{R}$ such that

$$V(\overline{x}) < V(x) \;\; \forall x \in X, \; \overline{x} \in f(x): \; x \notin X_\infty. \tag{2}$$

Thus, a Lyapunov function will strictly monotonically decrease along the trajectories of (1) until they reach a terminal state.

1) Termination in finite time:

Lemma 1: If the state space X is finite and if there exists a function V satisfying (2), then a terminal state X_∞ will be reached in a finite number of steps.

Proof: Denote $X_1 := X$ and $\widetilde{X}_1 := X_1 \setminus X_\infty$. Let $i = 1$ and define

$$x_{m_i} = \arg\min_{x \in \widetilde{X}_i} V(x) \tag{3}$$

Since $\widetilde{X}_1$ is finite, x_{m_1} exists and is well defined. Recursively define $X_{i+1} := X_i \setminus \{x_{m_i}\}$ and $\widetilde{X}_i := X_i \setminus X_\infty$ where x_{m_i} is defined according to (3). Since X is finite, only a finite number of sets X_i can be defined in this way and there exists $z \in \mathbb{Z}^+$ such that $X_{z+1} = \varnothing$. Now, let $\mathcal{S}$ denote the set of all sequences $\mathcal{X} = (x(0), x(1), \ldots, x(t), \ldots)$ of elements from X satisfying (1). Let $\mathcal{S}_1 := \mathcal{S}$ and denote by $\mathcal{S}_2$ the set of all sequences $\mathcal{X} \in \mathcal{S}_1$ for which $x(T) = x_{m_1}$ for some $T \in \mathbb{Z}^+$. Let $i = 1$ and until $\mathcal{S}_i \setminus \mathcal{S}_{i+1} = \varnothing$, recursively denote by $\mathcal{S}_{i+2}$ the set of all sequences $\mathcal{X} \in \mathcal{S}_i \setminus \mathcal{S}_{i+1}$ for which $x(T) = x_{m_i}$ for some $T \in \mathbb{Z}^+$. Now, assume that $\mathcal{X} \in \mathcal{S}_i$ and let $\overline{\overline{x}} \in f(x_{m_i})$. If $\overline{\overline{x}} \in \widetilde{X}$, then $V(\overline{\overline{x}}) < V(x_{m_i})$, which contradicts (3). Therefore, $\overline{\overline{x}} \in X_\infty$ and a terminal state has been reached. Proof is complete. ■

The following lemma provides a criterion that establishes finite-time termination in the general case that the state space X may perhaps be not finite.

Lemma 2: If there exists a bounded function $V: X \mapsto \mathbb{R}^-$, and a constant $\theta > 1$ satisfying

$$V(\overline{x}) < \theta V(x) \quad \forall x \in X, \; \overline{x} \in f(x): x \notin X_\infty. \tag{4}$$

then a terminal state X_∞ will be reached in a finite number of steps.

Proof: Since V is bounded, there exists $M \in \mathbb{R}^+$, such that $-M \leq V(x) < 0$, $\forall x \in X$. Now, assume that there exists a sequence $\mathcal{X} = (x(0), x(1), \ldots, x(t), \ldots)$ of elements from X satisfying (1) that does not reach a terminal state in finite time. I.e. $x(t) \notin X_\infty$, $\forall t \in \mathbb{Z}^+$. Then, $V(x(t)) < -M$ for

$$t > \frac{\log M - \log |V(x(0))|}{\log \theta}, \tag{5}$$

which contradicts bounded-ness of V. ■

The following Proposition is a theoretical result which may be used in proving finite-time termination of some processes. (See for instance Example 1 below.)

Proposition 1: Consider the dynamical system $\mathcal{S}(X, f, X_0, X_\infty)$ and let $\mathcal{S}$ denote the set of all sequences $\mathcal{X} = (x(0), x(1), \ldots, x(t), \ldots)$ of elements from X satisfying (1). A terminal state X_∞ will be reached in finite-time if and only if every sequential element $\mathcal{X}$ of $\mathcal{S}$ has a subsequence which reaches a terminal state in a finite number of steps.

2) Absence of overflow: As described earlier, absence of overflow can be characterized by avoidance of an unsafe subset X_{-} of the state space X. Lyapunov functions defined as in (2) can serve as the certificate for satisfaction of this property in the following way. Define the level sets $\mathcal{L}_r(V)$ of V, by

$$\mathcal{L}_r(V) = \{x \in X : \ V(x) < r\}$$

These level sets are invariant with respect to (1), in the sense that $x(t+1) \in \mathcal{L}_r(V)$ whenever $x(t) \in \mathcal{L}_r(V)$. We use this fact, along with the monotonicity property of V, to establish a separation between the reachable set and the unsafe region of the state space. We have the following proposition.

Proposition 2: Consider the system (1) and let $\mathcal{V}$ denote the space of all Lyapunov functions for this system satisfying (2). An unsafe subset X_{-} of the state space X can never be reached along all the trajectories of (1) if there exists $V \in \mathcal{V}$ satisfying

$$\inf_{x \in X_{-}} V(x) \geq \sup_{x \in X_0} V(x)$$

D. Storage functions as behavior certificates

A useful generalization of Lyapunov functions is given by the so-called *storage functions*. Assume that the set-valued function $f(x)$ is defined by

$$f(x) = \left\{\overline{f}(x,u) : u \in U\right\} \qquad (6)$$

where U is a given set and $\overline{f} : \ X \times U \mapsto X$ is a given function. In other words, assume that the uncertainty in the dynamics of $x = x(t)$ in (1) is caused by the presence of an external input $u = u(t) \in U$.

Definition 3: Let $V : \ X \mapsto \mathbb{R}$ and $\sigma : \ X \times U \mapsto \mathbb{R}$ be some functions. The function V is said to be a *storage function with supply rate σ* for system (1) if

$$V\left(\overline{f}(x,u)\right) - V(x) \leq \sigma(x,u) \quad \forall x \in X, \ u \in U. \qquad (7)$$

Storage functions can be very useful in the analysis of input/output behavior of systems. In particular, one could be interested in proving that system defined by (1) and (6) reaches a terminal state in finite time for *some* inputs $u = u(t)$, in a situation when it is known not to be true for an *arbitrary* input $u = u(t)$. Similarly absence of state overflow in the system (1),(6) may be proved for some inputs $u = u(t) \in \underline{U} \subset U$, while an arbitrary input $u(t) \in U$, may cause an overflow.

Example 1: Consider the system defined by (1) and (6), and assume that $\sigma(x,u) = -1$ for $u = 0$ and $\sigma(x,u) < 1$ for all u, x. In addition assume that the state space X is finite. Then, finite time termination for the input $u(t) = 0$ is obviously implied by (7) and Lemma (1). On the other hand, this storage function implies nothing about finite-time termination for inputs of the type $u(t) = c$, where c is a nonzero constant. A very interesting case arises when the inputs $u \in U$ are such that, on average, they are zero at least half of the time. Indeed, in this case too, finite-time termination for the system (1),(6) is guaranteed by (7). To show this, first notice that $\forall N_i \in \mathbb{Z}^+$, $\exists N_{i+1} > N_i$, such that:

$$\sum_{k=N_i}^{k=N_{i+1}} \sigma(x(k), u(k)) < 0 \qquad (8)$$

Now, from the sequence $\mathcal{X} = (x(0), x(1), ..., x(t), ...)$, extract the subsequence $\mathcal{X}_0 = (x(0), x(N_1), x(N_2), ...)$ where $N_1, N_2, ..$ are such that for each two consecutive elements, (8) is satisfied. Using (8) the elements of this sequence satisfy (2) and by Lemma (1), $\mathcal{X}_0$ reaches a terminal state in finite time. Employing Proposition 1, this proves that the system (1),(6) reaches a terminal state in finite time.

E. Convex optimization of Lyapunov functions

Our method of automated code analysis is based on using convex optimization in the search for Lyapunov functions certifying certain aspects of behavior of the dynamical system defined by a computer program.

The main difficulty in using Lyapunov functions in system analysis is finding them. The chances of finding a Lyapunov function successfully are increased when (2) is only required on a subset of $X \setminus X_\infty$. It is tempting to replace (2) with

$$V(\overline{x}) < V(x) \quad \forall x \in X,\ \overline{x} \in f(x): \\ V(x) < 1,\ x \notin X_\infty, \tag{9}$$

while adding the constraint

$$V(x) < 1 \quad \forall x \in X_0.$$

This way, V is not required to decrease monotonically for some of the states which cannot be reached from X_0. Unfortunately, in contrast with (2), formulation (9) has a significant disadvantage of being non-convex with respect to V. Thus, finding a solution of (9) is typically much harder than finding a solution of (2).

Alternatively, (2) can be replaced by

$$V(\overline{x}) < V(x) \quad \forall x \in X,\ \overline{x} \in f(x):\ x \in X_v,$$

where X_v is a *fixed* subset of X which does not contain any terminal points. Since V does not enter into any conditioning (of x) here, this set of constraints is convex with respect to V. This technique, along with partitioning of the state space into appropriate subspaces of smaller size, and assigning different functions V_i to each subspace X_{v_i} leads to systematic improvement of analysis.

At a basic level, the search for V involves selecting a finite dimensional linear parameterization

$$V(x) = V_\tau(x) = \sum_{k=1}^{k=n} \tau_k V_k(x), \\ \tau = (\tau_k)_{k=1}^N,\ \tau_k \in \mathbb{R} \tag{10}$$

of a Lyapunov function candidate V, where $V_k : X \mapsto \mathbb{R}$ are fixed functions, used as a basis of a linear subspace of the vector space of all real-valued functions $V : X \mapsto \mathbb{R}$. For every $\tau = (\tau_k)_{k=1}^N$ let

$$\phi(\tau) = \max_{x \in X, \overline{x} \in f(x):\ x \notin X_\infty} V_\tau(\overline{x}) - V_\tau(x)$$

(assuming for simplicity that the maximum does exist). Since ϕ is a maximum of a family of linear functions, ϕ is a convex function of its argument. If minimizing ϕ over the unit circle $|\tau| = 1$ yields a negative minimum, the optimal $\overline{\tau}$ defines a valid Lyapunov function $V_{\overline{\tau}}(x)$ for (2). Otherwise, no linear combination (10) yields a valid Lyapunov function for (2).

Efficiency of the proposed convex optimization approach depends heavily on computability of ϕ and its subgradients. While ϕ is convex with respect to its argument, the same does not necessarily hold for $V_\tau(\overline{x}) - V_\tau(x)$. In fact, even very simple computer codes lead to non-convex optimization as the problem to calculate the maximum of $V_\tau(\overline{x}) - V_\tau(x)$.

As the main tool for resolving this problem, we propose *convex relaxations*, which, in general, means replacing maximization of a function $h : Z \mapsto \mathbb{R}$ by maximization of the *linear* functional

$$\hat{h}(p_\xi) = \mathrm{E}h(\xi)$$

over an affine subset of the set of all probability distributions p_ξ of random variables ξ with values in Z. Various versions of the convex relaxations method, including using sums of squares in positivity verification, S-procedure lossless-ness in robustness analysis, and semidefinite relaxations of combinatorial problems, have already been used successfully in a number of applications.

III. SYSTEM INVARIANTS CERTIFYING BOUNDEDNESS AND/OR FINITE-TIME TERMINATION

Herein, we propose general forms for system invariants which establish the desired properties of computer programs and suitably fit in a convex optimization framework. Among several properties of a reliable software mentioned earlier, absence of overflow along with finite-time termination is desired in most applications.

Proposition 3: Consider the dynamical system $\mathcal{S} = \mathcal{S}(X, f, X_0, X_\infty)$ defined by (1) and assume that there exists a real-valued function $V : X \mapsto \mathbb{R}$ such that

$$V(\overline{x}) < \theta V(x) \quad \forall x \in X,\ \overline{x} \in f(x): x \notin X_\infty. \tag{11}$$

$$V(x) < 0 \quad \forall x \in X_0. \tag{12}$$

$$V(x) > \left\| \frac{x}{M} \right\|^2 - 1 \quad \forall x \in X. \tag{13}$$

where $\theta \in \mathbb{R}^+$ is a constant, and no constraint on finiteness of the state space X is imposed. Then, every solution $\mathcal{X} = x(t)$ of (1) remains bounded in the safe region defined by $|x_i| < M$, where each x_i is a component of the state vector x. Moreover, if $\theta > 1$, every solution $\mathcal{X} = x(t)$ reaches a terminal state X_∞ in finite time.

Proof: Note that (12) and (11) imply non-positivity of $V(x)$ on $X \setminus X_\infty$. Moreover, by (13), $V(x)$ is bounded from below by -1. Therefore, $V(x) \in (-1, 0)$. By Lemma 2, (11) implies finite-time termination. Also, the unsafe region X_- is defined by $|x_i| \geq M$. Therefore,

$$\left\| \frac{x}{M} \right\|^2 \geq 1, \ \forall x \in X_-,$$

$$\inf_{x \in X_-} V(x) = 0 \geq \ 0 = \sup_{x \in X_0} V(x)$$

Proposition 2 then completes the proof. ■

Remark 1: Appropriate choice of the constant θ, can determine the success or failure of the optimization algorithm to some extent. Larger values of θ in (11) imply/require faster convergence. This can be seen more explicitly in formulae (5) where the bound on the maximum number of iterations is proportional to the reciprocal of $\log \theta$. Intuitively, This means that larger values of θ are more restrictive and make it more difficult to satisfy $(11) - (13)$. Therefore, choosing smaller values of θ close to 1 is recommended for more successful analysis.

Particularly, by choosing $\theta < 1$, only boundedness will be established.

Remark 2: As mentioned in the Introduction, absence of 'array index out-of-bounds' calls is crucial in safety of software. Essentially, this property is equivalent to bounded-ness of some integer variables (array indices) in a much smaller region of the state space determined by the size of the array or matrix in general. Therefore, this property as well, can be verified by employing Proposition 3.

Remark 3: By imposing a quadratic form on V, the search for a Lyapunov-like function satisfying $(11) - (13)$ reduces to semidefinite programming [1]. Another possibility is to let V be a polynomial function of the state variables x_i. In this case, the search for system invariants restricted to the class of polynomials with real coefficients can be formulated as a sums of squares problem [9],[10].

IV. CONCLUSIONS

A novel approach towards analysis of real-time software was introduced. It was shown that software, as a rule for iterative modification of computer memory, can be modeled as a dynamical system. Specific models carrying this task were also suggested. System invariants, found by Lagrangian relaxations and convex optimization of certain Lyapunov-like functions prove the desired properties of the dynamical system/software. The properties include bounded-ness of all variables within acceptable ranges and finite time termination of the program in most cases. It was shown through a numerical example that the method works successfully. However, scalability of the technique needs to be improved for applications to large computer programs with thousands of lines of code.

REFERENCES

[1] S. Boyd, L.E. Ghaoui, E. Feron, and V. Balakrishnan. Linear Matrix Inequalities in Systems and Control Theory. Society for Industrial and Applied Mathematics, 1994.

[2] D. Bertsimas, and J. Tsitsikilis. Introduction to Linear Optimization. Athena Scientific, 1997.

[3] M. A. Colon, S. Sankaranarayanan, H. B. Sipma. Linear invariant generation using non-linear constraint solving. In Computer Aided Verification (CAV 2003), vol. 2725 of Lecture Notes in Computer Science, Springer Verlag, pp. 420-433.

[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proc. 4th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '77, pages 238–252, 1977.

[5] P. Cousot, and R. Cousot. Systematic design of program analysis frameworks. In Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York.

[6] P. Cousot. Semantic foundations of program analysis. In S. Muchnick and N. Jones, editors, Program Flow Analysis: Theory and Applications, chapter 10, pages 303–342. Prentice-Hall, 1981.

[7] D. Dams. Abstract interpretation and partition refinement for Model Checking. Ph.D. Thesis, Eindhoven University of Technology, 1996.

[8] D. Monniaux. Abstract interpretation of programs as Markov decision processes. In Static Analysis Symposium, volume 2694 in Lecture Notes in Computer Science, pages 237-254, Springer Verlag, 2003.

[9] P. A. Parrilo. Minimizing Polynomial Functions. In Algorithmic and Quantitative Real Algebraic Geometry, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 60, pp. 83–99, AMS.

[10] K. Gatermann, and P.A. Parrilo. Symmetry groups, semidefinite programs, and sums of squares. Journal of Pure and Appl. Algebra, Vol. 192, No. 1-3, pp. 95-128, 2004.