Robust Non-termination Analysis of Numerical Software

Bai Xue[1]([✉]), Naijun Zhan[1,2]([✉]), Yangjia Li[1,3]([✉]), and Qiuye Wang[1,2]([✉])

[1] State Key Laboratory of Computer Science, Institute of Software, CAS,
Beijing, China
{xuebai,znj,yangjia,wangqy}@ios.ac.cn
[2] University of Chinese Academy of Sciences, CAS, Beijing, China
[3] University of Tartu, Tartu, Estonia

Abstract. Numerical software is widely used in safety-critical systems such as aircrafts, satellites, car engines and many other fields, facilitating dynamics control of such systems in real time. It is therefore absolutely necessary to verify their correctness. Most of these verifications are conducted under ideal mathematical models, but their real executions may not follow the models exactly. Factors that are abstracted away in models such as rounding errors can change behaviors of systems essentially. As a result, guarantees of verified properties despite the present of disturbances are needed. In this paper, we attempt to address this issue of nontermination analysis of numerical software. Nontermination is often an unexpected behaviour of computer programs and may be problematic for applications such as real-time systems with hard deadlines. We propose a method for robust conditional nontermination analysis that can be used to under-approximate the maximal robust nontermination input set for a given program. Here robust nontermination input set is a set from which the program never terminates regardless of the aforementioned disturbances. Finally, several examples are given to illustrate our approach.

Keywords: Numerical software · Nontermination analysis · Robust verification

1 Introduction

Software is ubiquitous in mission-critical and safety-critical industrial infrastructures as it is, in principle, the most effective way to manipulate complex systems in real time. However, many computer scientists and engineers have experienced

costly bugs in embedded software. Examples include the failure of the Ariane 5.01 maiden flight (due to an overflow caused by an unprotected data conversion from a too large 64-bit floating point to a 16-bit signed integer value), the failure of the Patriot missile during the Gulf war (due to an accumulated rounding error), the loss of Mars orbiter (due to a unit error). Those examples indicate that mission-critical and safety-critical software may be far from being safe [11]. It is therefore absolutely necessary to prove the correctness of software by using formal, mathematical techniques that enable the development of correct and reliable software systems.

The dominant approach to the verification of programs is called *Floyd-Hoare-Naur inductive assertion approach* [13,18,34]. It uses *pre-* and *post-condition* to specify the condition of initial states and the property that should be satisfied by terminated states, and use *Hoare logic* to reason about properties of programs. The hardest parts of this approach are *invariant generation* and *termination analysis*. It is well-know that the termination or nontermination problem is undecidable, and even not semi-decidable in general. Thus, more practical approaches include present some sufficient conditions for termination, or some sufficient conditions for nontermination, or put these two types of conditions in parallel, or prove the decidability for some specific families of programs, e.g., [3,7,14,16,23,25,39,53].

On the other hand, most of these verifications are conducted under ideal mathematical models, but their real executions may not follow the models exactly. Factors that are abstracted away in models such as rounding errors can change behaviors of systems essentially. As a result, guarantees of verified properties despite the present of disturbances are needed. We notice that this problem affects most existing termination/nontermination as well.

In [54], the authors presented the following example:

Example 1. Consider a simple loop

$$\text{Q1:} \quad \textbf{while } (B\mathbf{x} > \mathbf{0}) \ \{\mathbf{x} := A\mathbf{x}\},$$

where $A = \begin{pmatrix} 2 & -3 \\ -1 & 2 \end{pmatrix}$, $B = \begin{pmatrix} 1 & b \\ -1 & b \end{pmatrix}$ with $b = -\frac{1127637245}{651041667} = -\sqrt{3} + \epsilon \sim -1.732050807$.

So we have $\epsilon = \sqrt{3} - (-\frac{1127637245}{651041667}) > 0$ a small positive number. Here we take 10 decimal digits of precision.

According to the termination decidability result on simple loops proved in [48], Q1 terminates based on exact computation. But unfortunately, it does not terminate in practice as the core decision procedure given in [48] invokes a procedure to compute *Jordan normal form* based on numeric computation for a given matrix, and thus the floating error has to be taken into account. In order to address this issue, a symbolic decision procedure was proposed in [54]. However, a more interesting and challenging issue is to find a systematic way to take all possible disturbances into account during conducting termination and non-termination analysis in practical numerical implementations.

In this paper we attempt to address this challenge, and propose a framework for robust nontermination analysis for numerical software based on control theory as in [40], which proposes a control-theoretic framework based on Lyapunov invariants to conduct verification of numerical software. Non-termination analysis proves that programs, or parts of a program, do not terminate. Nontermination is often an unexpected behaviour of computer programs and implies the presence of a bug. If a nonterminating computation occurs, it may be problematic for applications such as real-time systems with hard deadlines or situations when minimizing workload is important. In this paper, computer programs of interest are restricted to a class of computer programs composed of a single loop with a complicated switch-case type loop body. These programs can also be viewed as a constrained piecewise discrete-time dynamical system with time-varying uncertainties. We reformulate the problem of determining robust conditional nontermination as finding the maximal robust nontermination input set of the corresponding dynamical system, and characterize that set using a value function, which turns out to be a solution to a suitable mathematical equation. In addition, when the dynamics of the piecewise discrete-time system in each mode is polynomial and the state and uncertain input constraints are semi-algebraic, the optimal control problem is relaxed as a semi-definite programming problem, to which its polynomial solution forms an inner-approximation of the maximal robust nontermination input set when exists. Such relaxation is sound but incomplete. Finally, several examples are given to illustrate our approach.

It should be noticed that the concept of robust nontermination input sets is essentially equivalent to the maximal robustly positive invariants in control theory. Interested readers can refer to, e.g., [2,40,45,47]. Computing the maximal robustly positive invariant of a given dynamical system is still a long-standing and challenging problem not only in the community of control theory. Most existing works on this subject focus on linear systems, e.g. [21,38,45,47,51]. Although some methods have been proposed to synthesize positively invariants for nonlinear systems, e.g., the barrier certificate generation method as in [36,37] and the region of attraction generation method as in [15,19,30,49]. These methods, however, resort to bilinear sum-of-squares programs, which are notoriously hard to solve. In order to solve the bilinear sum-of-squares programs, a commonly used method is to employ some form of alteration (e.g., [19,30,50]) with a feasible initial solution to the bilinear sum-of-squares program. Recently, [43,44] proposed linear programming based methods to synthesize maximal (robustly) positive polyhedral invariants. Contrasting with aforementioned methods, in this paper we propose a semi-definite programming based method to compute semi-algebraic invariant. Our method does not require an initial feasible solution.

Organization of the Paper. The structure of this paper is as follows. In Sect. 2, basic notions used throughout this paper and the problem of interest are introduced. Then we elucidate our approach for performing conditional non-termination analysis in Sect. 3. After demonstrating our approach on several illustrating examples in Sect. 4, we discuss related work in Sect. 5 and finally conclude this paper in Sect. 6.

2 Preliminaries

In this section we describe the programs which are considered in this paper and we explain how to analyze them through their representation as piecewise discrete-time dynamical systems.

The following basic notations will be used throughout the rest of this paper: $\mathbb{N}$ stands for the set of nonnegative integers and $\mathbb{R}$ for the set of real numbers; $\mathbb{R}[\cdot]$ denotes the ring of polynomials in variables given by the argument, $\mathbb{R}_d[\cdot]$ denotes the vector space of real multivariate polynomials of degree d, $d \in \mathbb{N}$. Vectors are denoted by boldface letters.

2.1 Computer Programs of Interest

In this paper the computer program of interest, as described in Program 1, is composed of a single loop with a possibly complicated switch-case type loop body, in which variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ are assigned using parallel assignments $(x_1, \ldots, x_n) := \boldsymbol{f}(x_1, \ldots, x_n, d_1, \ldots, d_m)$, where $\boldsymbol{d} = (d_1, \ldots, d_m)$ is the vector of uncertain inputs, of which values are sampled nondeterministically from a compact set, i.e. $(d_1, \ldots, d_m) \in D$, such as round-off errors in performing computations. The form of programs under consideration is given in Program 1. In Program 1, $D = \{\boldsymbol{d} \mid \bigwedge_{i=1}^{n_{k+1}} h_{k+1,i}(\boldsymbol{d}) \leq 0\}$ is a compact set in $\mathbb{R}^m$ and $h_{k+1,i} : \mathbb{R}^m \mapsto \mathbb{R}$, is continuous over $\boldsymbol{d}$. $\Omega \subseteq \mathbb{R}^n$ stands for the initial condition on inputs; $X_0 = \{\boldsymbol{x} \in \mathbb{R}^n \mid \bigwedge_{i=1}^{n_0}[h_{0,i}(\boldsymbol{x}) \leq 0]\}$ stands for the loop condition, which is a compact set in $\mathbb{R}^n$; $X_j = \{\boldsymbol{x} \in \mathbb{R}^n \mid \bigwedge_{i=1}^{n_j}[h_{j,i}(\boldsymbol{x}) \rhd 0]\}$, $j = 1, \ldots, k$, stands for the j-th branch conditions, where $\rhd \in \{\leq, <\}$. $h_{j,i} : \mathbb{R}^n \mapsto \mathbb{R}$, $j = 0, \ldots, k$, $i = 1, \ldots, n_j$, $\boldsymbol{f}_l : \mathbb{R}^n \times D \mapsto \mathbb{R}^n$, $l = 1, \ldots, k$, are continuous functions over $\boldsymbol{x}$ and over $(\boldsymbol{x}, \boldsymbol{d})$ respectively. Moreover, $\{X_1, \ldots, X_k\}$ forms a complete partition of $\mathbb{R}^n$, i.e. $X_i \cap X_j = \emptyset$ for $\forall i \neq j$, where $i, j \in \{1, \ldots, k\}$, and $\cup_{j=1}^k X_j = \mathbb{R}^n$.

Program 1. Computer Programs of Interest

```
1  x := x₀;/* x₀ ∈ Ω                                                    */
2  while x ∈ X₀ do
          /* d ∈ D                                                      */
3      if x ∈ X₁ then
4      |   x := f₁(x, d);
5      end
6      else if x ∈ X₂ then
7      |   x := f₂(x, d);
8      end
9      . . .
10     else if x ∈ Xₖ then
11     |   x := fₖ(x, d);
12     end
13 end
```

As described in Program 1, an update of the variable $\boldsymbol{x}$ is executed by the i-th branch $\boldsymbol{f}_i : \mathbb{R}^n \times D \mapsto \mathbb{R}^n$ if and only if the current value of $\boldsymbol{x}$ satisfies the i-th branch condition X_i.

2.2 Piecewise Discrete-Time Systems

In this subsection we interpret Program 1 as a constrained piecewise discrete-time dynamical system with uncertain inputs. Formally,

Definition 1. *A constrained piecewise discrete-time dynamical system (PS) is a quintuple $(\boldsymbol{x}_0, X_0, \mathcal{X}, D, \mathcal{L})$ with*

- $\boldsymbol{x}_0 \in \Omega$ *is the condition on initial states;*
- $X_0 \subseteq \mathbb{R}^n$ *is the domain constraint, which is a compact set. A path can evolve complying with the discrete dynamics only if its current state is in X_0;*
- $\mathcal{X} := \{X_i, i = 1, \ldots, k\}$ *with X_i as interpreted in* Program 1*;*
- $D \subseteq \mathbb{R}^m$ *is the set of uncertain inputs;*
- $\mathcal{L} := \{\boldsymbol{f}_i(\boldsymbol{x}, \boldsymbol{d}), i = 1, \ldots, k\}$ *is the family of the continuous functions $\boldsymbol{f}_i(\boldsymbol{x}, \boldsymbol{d}) : X_i \times D \mapsto \mathbb{R}^n$.*

In order to enhance the understanding of PS, we use the following figure, i.e. Fig. 1, to illustrate it further. From now on, we associate a PS representation to each program of the form Program 1. Since a program may admit several PS representations, we choose one of them, but the choice does not change the results provided in this paper.
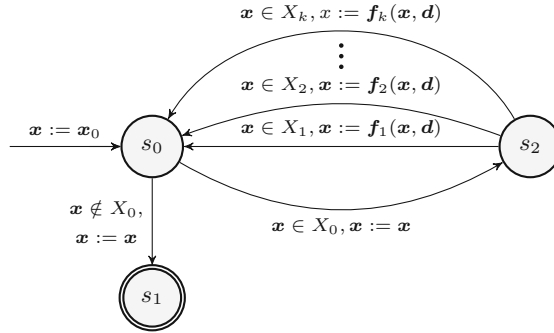
Fig. 1. An illustrating graph of PS

Definition 2. *An input policy π is an ordered sequence $\{\boldsymbol{\pi}(i), i \in \mathbb{N}\}$, where $\boldsymbol{\pi}(\cdot) : \mathbb{N} \mapsto D$, and Π is defined as the set of input policies, i.e. $\Pi = \{\pi \mid \boldsymbol{\pi}(\cdot) : \mathbb{N} \mapsto D\}$.*

If an input policy π makes Program 1 non-terminated from an initial state $\boldsymbol{x}_0$, then the trajectory $\boldsymbol{x}_{\boldsymbol{x}_0}^\pi : \mathbb{N} \mapsto \mathbb{R}^n$ from $\boldsymbol{x}_0$ following the discrete dynamics is defined by

$$\boldsymbol{x}_{\boldsymbol{x}_0}^\pi(l+1) = \boldsymbol{f}(\boldsymbol{x}_{\boldsymbol{x}_0}^\pi(l), \boldsymbol{\pi}(l)), \tag{1}$$

where $\boldsymbol{x}_{\boldsymbol{x}_0}^\pi(0) = \boldsymbol{x}_0$, $\forall l \in \mathbb{N}. \boldsymbol{x}_{\boldsymbol{x}_0}^\pi(l) \in X_0$, and

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{d}) = 1_{X_1} \cdot \boldsymbol{f}_1(\boldsymbol{x}, \boldsymbol{d}) + \cdots + 1_{X_k} \cdot \boldsymbol{f}_k(\boldsymbol{x}, \boldsymbol{d})$$

with $1_{X_i} : X_i \mapsto \{0, 1\}$, $i = 1, \ldots, k$, representing the indicator function of the set X_i, i.e.

$$1_{X_i} := \begin{cases} 1, & \text{if } \boldsymbol{x} \in X_i, \\ 0, & \text{if } \boldsymbol{x} \notin X_i. \end{cases}$$

Consequently, Program 1 is said to be robust nontermination starting from an initial state $\boldsymbol{x}_0 \in \Omega$ if for any input policy $\pi \in \Pi$, $\forall l \in \mathbb{N}. \boldsymbol{x}_{\boldsymbol{x}_0}^\pi(l) \in X_0$ holds. Formally,

Definition 3. *A program of* Program 1 *is said to be robust non-terminating w.r.t. an initial state* $\boldsymbol{x}_0 \in X_0$, *if*

$$\forall \pi \in \Pi. \ \forall l \in \mathbb{N}. \ \boldsymbol{x}_{\boldsymbol{x}_0}^\pi(l) \in X_0. \tag{2}$$

Now, we define our problem of deciding a set of initial states rendering Program 1 robust non-termination.

Definition 4 (Robust Nontermination Set). *A set* Ω *of initial states in* $\mathbb{R}^n$ *is a robust nontermination set for a program* P *of the form* Program 1 *if* P *is robustly non-terminating w.r.t.* $\boldsymbol{x}_0$ *for any* $\boldsymbol{x}_0 \in \Omega$. *We call* $\{\boldsymbol{x}_0 \in \mathbb{R}^n \mid P$ *is robustly non-terminating w.r.t.* $\boldsymbol{x}_0\}$ *the maximal robust non-termination set, denoted by* $\mathcal{R}_0$.

From Definition 4, we observe that $\mathcal{R}_0$ is a subset of X_0 such that all runs of Program 1 starting from it can not breach it forever, i.e. if $\boldsymbol{x}_0 \in \mathcal{R}_0$, $\boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{d}) \in \mathcal{R}_0$ for $\forall \boldsymbol{d} \in D$. Therefore, the set $\mathcal{R}_0$ is equivalent to the maximal robust positively invariant for PS (1) in control theory. For the formal concept of maximal robust positively invariant, please refer to, e.g., [2, 45, 47].

3   Robust Non-termination Set Generation

In this section we elucidate our approach of addressing the problem of robust conditional nontermination for Program 1, i.e. synthesizing robust non-termination sets as presented in Definition 4. For this sake, we firstly in Subsect. 3.1 characterize the maximal robust non-termination set $\mathcal{R}_0$ by means of the value function, which is a solution to a mathematical equation. Any solution to this optimal control problem generates a robust non-termination set. Then, in the case that $\boldsymbol{f}_i$, $i = 1, \ldots, k$, is polynomial over $\boldsymbol{x}$ and $\boldsymbol{d}$, and the constraint sets over $\boldsymbol{x}$ and $\boldsymbol{d}$, i.e. X_j, $j = 0, \ldots, k$, and D, are of the basic semi-algebraic form, the semidefinite program arising from sum-of-squares decompositions facilitates the gain of inner-approximations Ω of $\mathcal{R}_0$ via solving the relaxation of the derived optimal control problem in Subsect. 3.2.

3.1   Characterization of $\mathcal{R}_0$

In this subsection, we firstly introduce the value function to characterize the maximal robust nontermination set $\mathcal{R}_0$ and then formulate it as a solution to a constrained optimal control problem.

For $\boldsymbol{x}_0 \in \mathbb{R}^n$, the value function $V : \mathbb{R}^n \mapsto \mathbb{R}$ is defined by:

$$V(\boldsymbol{x}_0) := \sup_{\pi \in \Pi} \sup_{l \in \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} \{h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0}^\pi(l))\}. \tag{3}$$

Note that $V(\boldsymbol{x}_0)$ may be neither continuous nor semi-continuous. (A function $V' : X' \mapsto \mathbb{R}$ is lower semicontinuous iff for any $y \in \mathbb{R}$, $\{\boldsymbol{x} \in X' \mid V'(\boldsymbol{x}) \geq y\}$ is open, e.g., [4].)

The following theorem shows the relation between the value function V and the maximal robust nontermination set $\mathcal{R}_0$, that is, the zero sublevel set of $V(\boldsymbol{x}_0)$ is equal to the maximal robust nontermination set $\mathcal{R}_0$.

Theorem 1. $\mathcal{R}_0 = \{\boldsymbol{x}_0 \in \mathbb{R}^n \mid V(\boldsymbol{x}_0) \leq 0\}$, where $\mathcal{R}_0$ is the maximal robust nontermination set as in Definition 4.

Proof. Let $\boldsymbol{y}_0 \in \mathcal{R}_0$. According to Definition 4, we have that

$$\forall i \in \mathbb{N}. \ \forall \pi \in \Pi. \ \forall j \in \{1,\ldots,n_0\}. \ h_{0,j}(\boldsymbol{x}_{\boldsymbol{y}_0}^\pi(i)) \leq 0 \tag{4}$$

holds. Therefore, $V(\boldsymbol{y}_0) \leq 0$ and thus $\boldsymbol{y}_0 \in \{\boldsymbol{x}_0 \mid V(\boldsymbol{x}_0) \leq 0\}$.

On the other side, if $\boldsymbol{y}_0 \in \{\boldsymbol{x}_0 \in \mathbb{R}^n \mid V(\boldsymbol{x}_0) \leq 0\}$, then $V(\boldsymbol{y}_0) \leq 0$, implying that (4) holds. Therefore, $\boldsymbol{y}_0 \in \mathcal{R}_0$.

This concludes that $\mathcal{R}_0 = \{\boldsymbol{x}_0 \in \mathbb{R}^n \mid V(\boldsymbol{x}_0) \leq 0\}$.

From Theorem 1, the maximal robust nontermination set $\mathcal{R}_0$ could be constructed by computing $V(\boldsymbol{x}_0)$, which satisfies the dynamic programming principle as presented in Lemma 1.

Lemma 1. For $\forall \boldsymbol{x}_0 \in \mathbb{R}^n$ and $\forall l \in \mathbb{N}$, we have:

$$V(\boldsymbol{x}_0) = \sup_{\pi \in \Pi} \max \left\{ V(\boldsymbol{x}_{\boldsymbol{x}_0}^\pi(l)), \sup_{i \in [0,l) \cap \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0}^\pi(i)) \right\}. \tag{5}$$

Proof. Let

$$W(l, \boldsymbol{x}_0) := \sup_{\pi \in \Pi} \max \left\{ V(\boldsymbol{x}_{\boldsymbol{x}_0}^\pi(l)), \sup_{i \in [0,l) \cap \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0}^\pi(i)) \right\}. \tag{6}$$

We will prove that for $\epsilon > 0$, $|W(l, \boldsymbol{x}_0) - V(\boldsymbol{x}_0)| < \epsilon$.

According to the definition of $V(\boldsymbol{x}_0)$, i.e. (3), for any ϵ_1, there exists an input policy π' such that

$$V(\boldsymbol{x}_0) \leq \sup_{i \in \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} \{h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0}^{\pi'}(i))\} + \epsilon_1.$$

We then introduce two infinite uncertain input policies π_1 and π_2 such that $\pi_1 = \{\boldsymbol{\pi}_1(i), i \in \mathbb{N}\}$ with $\boldsymbol{\pi}_1(j) = \boldsymbol{\pi}'(j)$ for $j = 0, \ldots, l-1$ and $\pi_2 = \{\boldsymbol{\pi}_2(i), i \in \mathbb{N}\}$ with $\boldsymbol{\pi}(j) = \boldsymbol{\pi}'(j+l) \; \forall j \in \mathbb{N}$. Now, let $\boldsymbol{y} \in \boldsymbol{x}_{\boldsymbol{x}_0}^{\pi_1}(l)$, then we obtain that

$$W(l, \boldsymbol{x}_0) \geq \max \big\{ V(\boldsymbol{y}), \sup_{i \in [0,l) \cap \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} h_{0,j}(\boldsymbol{x}_{\boldsymbol{y}}^{\pi_1}(i)) \big\}$$

$$\geq \max \big\{ \sup_{i \in [l,+\infty) \cap \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} \{ h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0}^{\pi_2}(i-l)) \}, \sup_{i \in [0,l) \cap \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} \{ h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0}^{\pi_1}(i)) \} \big\}$$

$$= \max \big\{ \sup_{i \in [l,+\infty) \cap \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} \{ h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0}^{\pi'}(i)) \}, \sup_{i \in [0,l) \cap \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} \{ h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0}^{\pi'}(i)) \} \big\}$$

$$= \sup_{i \in \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} \{ h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0}^{\pi'}(i)) \}$$

$$\geq V(\boldsymbol{x}_0) - \epsilon_1.$$

Therefore,

$$V(\boldsymbol{x}_0) \leq W(l, \boldsymbol{x}_0) + \epsilon_1. \tag{7}$$

On the other hand, for any $\epsilon_1 > 0$, there exists a $\pi_1 \in \Pi$ such that $W(l, \boldsymbol{x}_0) \leq \max \big\{ V(\boldsymbol{x}_{\boldsymbol{x}_0}^{\pi_1}(l)), \sup_{i \in [0,l) \cap \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} \{ h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0}^{\pi_1}(i)) \} \big\} + \epsilon_1$, by the definition of $W(l, \boldsymbol{x}_0)$. Also, by the definition of $V(\boldsymbol{x}_0)$, i.e. (3), for any $\epsilon_1 > 0$, there exists a π_2 such that

$$V(\boldsymbol{y}) \leq \sup_{i \in \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} \{ h_{0,j}(\boldsymbol{x}_{\boldsymbol{y}}^{\pi_2}(i)) \} + \epsilon_1,$$

where $\boldsymbol{y} = \boldsymbol{x}_{\boldsymbol{x}_0}^{\pi_1}(l)$. We define $\pi \in \Pi$ such that $\boldsymbol{\pi}(i) = \boldsymbol{\pi}_1(i)$ for $i = 0, \ldots, l-1$ and $\boldsymbol{\pi}(i+l) = \boldsymbol{\pi}_2(i)$ for $\forall i \in \mathbb{N}$. Then, it follows

$$W(l, \boldsymbol{x}_0) \leq 2\epsilon_1 + \max \big\{ \sup_{i \in \mathbb{N} \cap [l,\infty)} \max_{j \in \{1,\ldots,n_0\}} \{ h_{0,j}(\boldsymbol{x}_{\boldsymbol{y}}^{\pi_2}(i-l)) \},$$

$$\sup_{i \in [0,l) \cap \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} \{ h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0}^{\pi_1}(i)) \} \big\} \tag{8}$$

$$\leq \sup_{i \in [0,+\infty) \cap \mathbb{N}} \max_{j \in \{1,\ldots,n_0\}} \{ h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0}^{\pi}(i)) \} + 2\epsilon_1$$

$$\leq V(\boldsymbol{x}_0) + 2\epsilon_1.$$

Combining (7) and (8), we finally have $|V(\boldsymbol{x}_0) - W(l, \boldsymbol{x}_0)| \leq \epsilon = 2\epsilon_1$, implying that $V(\boldsymbol{x}_0) = W(l, \boldsymbol{x}_0)$ since ϵ_1 is arbitrary. This completes the proof.

Based on Lemma 1 stating that the value function $V(\boldsymbol{x}_0)$ complies with the dynamic programming principle (5), we derive a central equation of this paper, which is formulated formally in Theorem 2.

Theorem 2. *The value function $V(\boldsymbol{x}_0) : \mathbb{R}^n \mapsto \mathbb{R}$ in (3) is a solution to the equation*

$$\min \big\{ \inf_{\boldsymbol{d} \in D} (V(\boldsymbol{x}_0) - V(\boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{d}))), V(\boldsymbol{x}_0) - \max_{j \in \{1,\ldots,n_0\}} h_{0,j}(\boldsymbol{x}_0) \big\} = 0. \tag{9}$$

Proof. It is evident that (9) is derived from (5) when $l = 1$.

According to Theorem 2, we conclude that *if there does not exist a solution to* (9), *the robust nontermination set* $\mathcal{R}_0$ *is empty.* Moreover, according to Theorem 2, $V(\boldsymbol{x}_0)$ as defined in (3) is a solution to (9). Note that the solution to (9) may be not unique, and we do not go deeper into this matter in this paper. However, any solution to (9) forms an inner-approximation of the maximal robust nontermination set, as stated in Corollary 1.

Corollary 1. *For any function* $u(\boldsymbol{x}_0) : \mathbb{R}^n \mapsto \mathbb{R}$ *satisfying* (9), $\{\boldsymbol{x}_0 \in \mathbb{R}^n \mid u(\boldsymbol{x}_0) \leq 0\}$ *is an inner-approximation of the maximal robust nontermination set* $\mathcal{R}_0$, *i.e.* $\{\boldsymbol{x}_0 \in \mathbb{R}^n \mid u(\boldsymbol{x}_0) \leq 0\} \subset \mathcal{R}_0$.

Proof. Let $u(\boldsymbol{x}_0) : \mathbb{R}^n \mapsto \mathbb{R}$ be a solution to (9). It is evident that $u(\boldsymbol{x}_0)$ satisfies the constraints:

$$\begin{cases} u(\boldsymbol{x}_0) - u(\boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{d})) \geq 0, \ \forall \boldsymbol{x}_0 \in \mathbb{R}^n, \forall \boldsymbol{d} \in D, \\ u(\boldsymbol{x}_0) - h_{0,j}(\boldsymbol{x}_0) \geq 0, \qquad \forall \boldsymbol{x}_0 \in \mathbb{R}^n, \forall j \in \{1, \ldots, n_0\} \end{cases} \tag{10}$$

Assume $\boldsymbol{x}_0' \in \{\boldsymbol{x}_0 \mid u(\boldsymbol{x}_0) \leq 0\}$. According to (10), we have that for $\forall \pi \in \Pi$, $\forall l \in \mathbb{N}$ and $\forall j \in \{1, \ldots, n_0\}$,

$$\begin{cases} u(\boldsymbol{x}_{\boldsymbol{x}_0'}^\pi(l+1)) & \leq u(\boldsymbol{x}_{\boldsymbol{x}_0'}^\pi(l)) \leq u(\boldsymbol{x}_0') \\ h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0'}^\pi(l)) & \leq u(\boldsymbol{x}_{\boldsymbol{x}_0'}^\pi(l)) \leq u(\boldsymbol{x}_0') \end{cases}. \tag{11}$$

Therefore, $\sup_{l \in \mathbb{N}} \max_{j \in \{1, \ldots, n_0\}} \{h_{0,j}(\boldsymbol{x}_{\boldsymbol{x}_0'}^\pi(l))\} \leq u(\boldsymbol{x}_0') \leq 0$, implying that $\boldsymbol{x}_0' \in \mathcal{R}_0$. Thus, $\{\boldsymbol{x}_0 \in \mathbb{R}^n \mid u(\boldsymbol{x}_0) \leq 0\} \subset \mathcal{R}_0$.

From Corollary 1, it is clear that an approximation of $\mathcal{R}_0$ from inside, i.e. a robust nontermination set, is able to be constructed by addressing (9). The solution to (9) could be addressed by grid-based numerical methods such as level set methods [12,32], which are a popular method for interface capturing. Such grid-based methods are prohibitive for systems of dimension greater than four without relying upon specific system structure. Besides, we observe that a robust nontermination set could be searched by solving (10) rather than (9). In the subsection that follows we relax (10) as a sum-of-squares decomposition problem in a semidefinite programming formulation when in Program 1, $\boldsymbol{f}_i$s are polynomials over $\boldsymbol{x}$ and $\boldsymbol{d}$, state and uncertain input constraints, i.e. X_js and Ds, are restricted to basic semi-algebraic sets.

3.2 Semi-definite Programming Implementation

In practice, it is non-trivial to obtain a solution $V(\boldsymbol{x}_0)$ to (2), and thus non-trivial to gain $\mathcal{R}_0$. In this subsection, thanks to (10) and Corollary 1, we present a semidefinite programming based method to solve (9) approximately and construct a robust invariant Ω as presented in Definition 4 when Assumption 1 holds.

Assumption 1. $\boldsymbol{f}_i$, $i = 1, \ldots, k$, *is polynomial over* $\boldsymbol{x}$ *and* $\boldsymbol{d}$, X_j *and* D, $j = 0, \ldots, k$, *are restricted to basic semi-algebraic sets in* Program 1.

Firstly, (10) has indicator functions on the expression $u(\boldsymbol{x}_0) - u(\boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{d}))$, which is beyond the capability of the solvers we use. We would like to obtain a constraint by removing indicators according to Lemma 2.

Lemma 2 ([8]). *Suppose $\boldsymbol{f}'(\boldsymbol{x}) = 1_{F_1} \cdot \boldsymbol{f}'_1(\boldsymbol{x}) + \cdots + 1_{F_{k'}} \cdot \boldsymbol{f}'_{k'}(\boldsymbol{x})$ and $\boldsymbol{g}'(\boldsymbol{x}) = 1_{G_1} \cdot \boldsymbol{g}'_1(\boldsymbol{x}) + \cdots + 1_{G_{l'}} \cdot \boldsymbol{g}'_{l'}(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{R}^n$, $k', l' \in \mathbb{N}$, and $F_i, G_j \subseteq \mathbb{R}^n$, $i = 1, \dots, k'$, $j = 1, \dots, l'$. Also, $F_1, \dots, F_{k'}$ and $G_1, \dots, G_{l'}$ are respectively disjoint. Then, $\boldsymbol{f}' \leq \boldsymbol{g}'$ if and only if (pointwise)*

$$
\bigwedge_{i=1}^{k'} \bigwedge_{j=1}^{l'} \left[F_i \wedge G_j \Rightarrow \boldsymbol{f}'_i \leq \boldsymbol{g}'_j \right] \wedge
$$
$$
\bigwedge_{i=1}^{k'} \left[F_i \wedge \left(\bigwedge_{j=1}^{l'} \neg G_j \right) \Rightarrow \boldsymbol{f}'_i \leq 0 \right] \wedge \tag{12}
$$
$$
\bigwedge_{j=1}^{l'} \left[\left(\bigwedge_{i=1}^{k'} \neg F_i \right) \wedge G_j \Rightarrow 0 \leq \boldsymbol{g}'_j \right].
$$

Consequently, according to Lemma 2, the equivalent constraint without indicator functions of (10) is equivalently formulated below:

$$
\bigwedge_{i=1}^{k} \left[\forall \boldsymbol{d} \in D. \ \forall \boldsymbol{x}_0 \in X_i. \ u(\boldsymbol{x}_0) - u(\boldsymbol{f}_i(\boldsymbol{x}_0, \boldsymbol{d})) \geq 0 \right] \wedge
$$
$$
\bigwedge_{j=1}^{n_0} \left[\forall \boldsymbol{x}_0 \in \mathbb{R}^n. \ u(\boldsymbol{x}_0) - h_{0,j}(\boldsymbol{x}_0) \geq 0 \right]. \tag{13}
$$

Before encoding (13) in sum-of-squares programming formulation, we denote the set of sum of squares polynomials over variables $\boldsymbol{y}$ by $\text{SOS}(\boldsymbol{y})$, i.e.

$$
\text{SOS}(\boldsymbol{y}) := \{ p \in \mathbb{R}[\boldsymbol{y}] \mid p = \sum_{i=1}^{r} q_i^2, q_i \in \mathbb{R}[\boldsymbol{y}], i = 1, \dots, r \}.
$$

Besides, we define the set $\Omega(X_0)$ of states being reachable from the set X_0 within one step computation, i.e.,

$$
\Omega(X_0) := \{ \boldsymbol{x} \mid \boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{d}), \boldsymbol{x}_0 \in X_0, \boldsymbol{d} \in D \} \cup X_0, \tag{14}
$$

which can be obtained by semi-definite programming or linear programming methods as in [24, 31]. Herein, we assume that it was already given. Consequently, when Assumption 1 holds and $u(\boldsymbol{x})$ in (13) is constrained to polynomial type and is restricted in a ball $B = \{ \boldsymbol{x} \mid h(\boldsymbol{x}) \geq 0 \}$, where $h(\boldsymbol{x}) = R - \sum_{i=1}^{n} x_i^2$ and $\Omega(X_0) \subseteq B$, (13) is relaxed as the following sum-of-squares programming problem:

$$\min_{u, s_{i,l_1}^{X_i}, s_{i,l_2}^{D}, s_{i,l}, s_{1,j}'} \boldsymbol{c}' \cdot \boldsymbol{w}$$

$$u(\boldsymbol{x}) - u(\boldsymbol{f}_i(\boldsymbol{x}, \boldsymbol{d})) + \sum_{l_1=1}^{n_i} s_{i,l_1}^{X_i} h_{i,l_1}(\boldsymbol{x}) + \sum_{l_2=1}^{n_{k+1}} s_{i,l_2}^{D} h_{k+1,l}(\boldsymbol{d}) - s_{i,1} h(\boldsymbol{x}) \in \texttt{SOS}(\boldsymbol{x}, \boldsymbol{d}),$$

$$u(\boldsymbol{x}) - h_{0,j}(\boldsymbol{x}) - s_{1,j}' h(\boldsymbol{x}) \in \texttt{SOS}(\boldsymbol{x}),$$

$$i = 1, \ldots, k; j = 1, \ldots, n_0,$$

(15)

where $\boldsymbol{c}' \cdot \boldsymbol{w} = \int_B u d\mu(\boldsymbol{x})$, $\boldsymbol{w}$ is the vector of the moments of the Lebesgue measure over B indexded in the same basis in which the polynomial $u(\boldsymbol{x}) \in \mathbb{R}_d[\boldsymbol{x}]$ with coefficients $\boldsymbol{c}$ is expressed, $s_{i,l_1}^{X_i}, s_{i,l_2}^{D}, s_{i,1} \in \texttt{SOS}(\boldsymbol{x}, \boldsymbol{d})$, $i = 1, \ldots, k$, $l_1 = 1, \ldots, n_i$, $l_2 = 1, \ldots, n_{k+1}$, $s_{1,j}' \in \texttt{SOS}(\boldsymbol{x})$, $j = 1, \ldots, n_0$, are sum-of-squares polynomials of appropriate degree. The constraints that polynomials are sum-of-squares can be written explicitly as linear matrix inequalities, and the objective is linear in the coefficients of the polynomial $u(\boldsymbol{x})$; therefore problem (15) is reformulated as an semi-definite program, which falls within the convex programming framework and can be solved via interior-points method in polynomial time (e.g., [52]). Note that the objective of (15) facilitate the gain of the less conservative robust nontermination set.

The implementation based on the sum-of-squares program (15) is sound but incomplete. Its soundness is presented in Theorem 3.

Theorem 3 (Soundness). *Let $u(\boldsymbol{x}) \in \mathbb{R}_d[\boldsymbol{x}]$ be solution to (15), then $\{\boldsymbol{x} \in B \mid u(\boldsymbol{x}) \leq 0\}$ is an inner-approximation of $\mathcal{R}_0$, i.e., every possible run of Program 1 starting from a state in $\{\boldsymbol{x} \in B \mid u(\boldsymbol{x}) \leq 0\}$ does not terminate.*

Proof. Since $u(\boldsymbol{x})$ satisfies the constraint in (15), we obtain that $u(\boldsymbol{x})$ satisfies according to $\mathcal{S}-$ procedure in [5]:

$$\bigwedge_{i=1}^{k} \left[\forall \boldsymbol{d} \in D. \ \forall \boldsymbol{x} \in X_i \cap B. \ u(\boldsymbol{x}) - u(\boldsymbol{f}_i(\boldsymbol{x}, \boldsymbol{d})) \geq 0 \right] \wedge \tag{16}$$

$$\bigwedge_{j=1}^{n_0} \left[\forall \boldsymbol{x} \in B. \ u(\boldsymbol{x}) - h_{0,j}(\boldsymbol{x}) \geq 0 \right]. \tag{17}$$

Due to (16) and the fact that $\cup_{i=1}^{k} X_i = \mathbb{R}^n$, we obtain that for $\forall \boldsymbol{x}_0 \in \{\boldsymbol{x} \in B \mid u(\boldsymbol{x}) \leq 0\}$, $\exists i \in \{1, \ldots, k\}. \ \forall \boldsymbol{d} \in D. \ u(\boldsymbol{x}_0) - u(\boldsymbol{f}_i(\boldsymbol{x}_0, \boldsymbol{d})) \geq 0$, implying that

$$u(\boldsymbol{x}_0) - u(\boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{d})) \geq 0, \forall \boldsymbol{d} \in D. \tag{18}$$

Assume that there exist an initial state $\boldsymbol{y}_0 \in \{\boldsymbol{x} \in B \mid u(\boldsymbol{x}) \leq 0\}$ and an input policy π' such that $\boldsymbol{x}_{\boldsymbol{y}_0}^{\pi'}(l) \in X_0$ does not hold for $\forall l \in \mathbb{N}$. Due to the fact that (17) holds, we have the conclusion that $\{\boldsymbol{x} \in B \mid u(\boldsymbol{x}) \leq 0\} \subset X_0$ and thus $\boldsymbol{y}_0 \in X_0$. Let $l_0 \in \mathbb{N}$ be the first time making $\boldsymbol{x}_{\boldsymbol{y}_0}^{\pi'}(l)$ violate the constraint X_0, i.e., $\boldsymbol{x}_{\boldsymbol{y}_0}^{\pi'}(l_0) \notin X_0$ and $\boldsymbol{x}_{\boldsymbol{y}_0}^{\pi'}(l) \in X_0$ for $l = 0, \ldots, l_0 - 1$. Also, since $\Omega(X_0) \subset B$, (18) and (17), where $\Omega(X_0)$ is defined in (14), we derive that $\boldsymbol{x}_{\boldsymbol{y}_0}^{\pi'}(l_0 - 1) \in \{\boldsymbol{x} \in B \mid u(\boldsymbol{x}) \leq 0\}$ and $u(\boldsymbol{x}_{\boldsymbol{y}_0}^{\pi}(l_0)) > 0$, which contradicts (18).

Thus, every possible run of Program 1 initialized in $\{x \in B \mid u(x) \leq 0\}$ will live in $\{x \in B \mid u(x) \leq 0\}$ forever while respecting X_0.

Therefore, the conclusion in Theorem 3 is justified.

4   Experiments

In this section we evaluate the performance of our method built upon the semi-definite program (15). Examples 2 and 3 are constructed to illustrate the soundness of our method. Example 4 is used to evaluate the scalability of our method in dealing with Program 1. The parameters that control the performance of our approach in applying (15) to these three examples are presented in Table 1. All computations were performed on an i7-7500U 2.70GHz CPU with 32GB RAM running Windows 10. For numerical implementation, we formulate the sum of squares problem (15) using the MATLAB package YALMIP[1] [29] and use Mosek[2] [33] as a semi-definite programming solver.

Table 1. Parameters and performance of our implementations on the examples presented in this section. $d_u, d_{s_{i,l_1}^{X_i}}, d_{s_{i,l_2}^{D}}, d_{s_{i,l}}, d_{s'_{1,j}}$: the degree of the polynomials $u, s_{i,l_1}^{X_i}, s_{i,l_2}^{D}, s_{i,l}, s'_{1,j}$ in (15), respectively, $i = 1, \ldots, k$, $l_1 = 1, \ldots, n_i$, $l_2 = 1, \ldots, n_{k+1}$, $j = 1, \ldots, n_0$; $Time$: computation times (seconds).

Ex.	d_h	$d_{s_{i,l_1}^{X_i}}$	$d_{s_{i,l_2}^{D}}$	$d_{s_{i,l}}$	$d_{s'_{1,j}}$	$Time$
1	14	14	14	14	14	11.30
1	16	16	16	16	16	28.59
2	6	12	12	12	6	9.06
2	8	16	16	16	8	65.22
2	10	20	20	20	10	123.95
2	12	24	24	24	12	623.95
4	4	4	4	4	4	58.56
4	5	4	4	4	4	60.02

Example 2. This simple example is mainly constructed to illustrate the difference between Program 1 taking uncertain inputs into account and free of disturbances. In both cases, Program 1 is composed of a single loop without switch-case type in loop body, i.e. $k = 1$ and $X_1 = \mathbb{R}^2$.

In case that $f_1(x, y) = (0.4x + 0.6y; dx + 0.9y)$, $X_0 = \{(x, y) \mid x^2 + y^2 - 1 \leq 0\}$ and $D = \{d \mid d^2 - 0.01 \leq 0\}$ in Program 1, the inner-approximations of the

---

[1] It can be downloaded from https://yalmip.github.io/.
[2] For academic use, the software Mosek can be obtained free from https://www.mosek.com/.

maximal robust nontermination set $\mathcal{R}_0$ are illustrated in Fig. 2(Left) when $d_u = 10$ and $d_u = 12$. By visualizing the results in Fig. 2, the inner-approximation obtained when $d_u = 12$ does not improve the one when $d_u = 10$ a lot. Although there is a gap between the inner-approximations obtained via our method and the set $\mathcal{R}_0$, it is not big.

In the ideal implementation of Program 1, that is, d in the loop body is a fixed nominal value, there will exists some initial conditions such that Program 1 in the real implementation may violate the constraint set X_0, i.e. Program 1 may terminate. We use $d = 0$ as an instance to illustrate such situation. The difference between termination sets is visualized in Fig. 2(Right). The robust nontermination set in case of $d \in [-0.1, 0.1]$ is smaller than the nontermination set when $d = 0$. Note that from Fig. 3, we observe that the inner-approximation obtained by our method when $d_u = 10$ can approximate $\mathcal{R}_0$ very well.
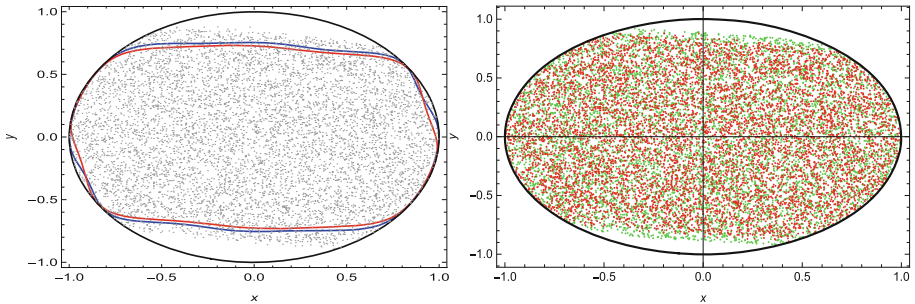
Fig. 2. Computed robust nontermination sets for Example 2. Left: (Blue and Red curves – the boundaries of the computed robust nontermination set $\mathcal{R}_0$ when $d_u = 10$ and $d_u = 12$, respectively; Gray points – the approximated robust nontermination set via numerical simulation techniques; Black curve – the boundary of X_0.) Right: (Green and red points – the approximated (robust) nontermination sets via numerical simulation techniques for Program 1 without and with disturbance inputs, respectively; Black curve – the boundary of X_0.) (Color figure online)

Example 3. In this example we consider Program 1 with switch-case type in the loop body, where $\boldsymbol{f}_1(x, y) = (x; (0.5 + d)x - 0.1y)$, $\boldsymbol{f}_2(x, y) = (y; 0.2x - (0.1 + d)y + y^2)$, $X_0 = \{(x, y) \mid x^2 + y^2 - 0.8 \leq 0\}$, $X_1 = \{(x, y) \mid 1 - (x - 1)^2 - y^2 \geq 0\}$, $X_2 = \{(x, y) \mid -1 + (x - 1)^2 + y^2 < 0\}$ and $D = \{d \mid d^2 - 0.01 \leq 0\}$. The inner-approximations computed by solving (15) when $d_u = 8, 10$ and 12 respectively are illustrated in Fig. 4. By comparing these results, we observe that polynomials of higher degree facilitate the gain of less conservative estimation of the set $\mathcal{R}_0$.

Example 4. In this example, we consider Program 1 with seven variables $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ and illustrate the scalability of our approach. In Program 1, $\boldsymbol{f}_1(\boldsymbol{x}) = ((0.5 + d)x_1; 0.8x_2; 0.6x_3 + 0.1x_6; x_4; 0.8x_5; 0.1x_2 + x_6; 0.2x_2 +$
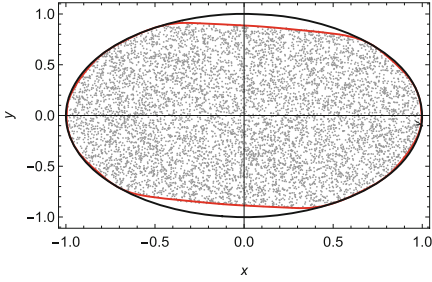
Fig. 3. Nontermination set estimation for Example 2. (Black and Red curves: the boundaries of X_0 and the computed robust nontermination set $\mathcal{R}_0$ when $d_u = 16$, respectively; Gray points – the approximated robust nontermination set via numerical simulation techniques.) (Color figure online)
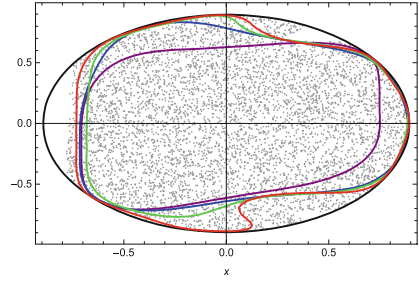
Fig. 4. Robust nontermination sets for Example 3. Black, Purple, Blue, Green and Red curves: the boundaries of X_0 and the computed robust nontermination sets $\mathcal{R}_0$ when $d_u = 6, 8, 10, 12$, respectively; Gray points – the approximated robust nontermination set via numerical simulation techniques. (Color figure online)

$0.6x_7)$;, $\boldsymbol{f}_2(\boldsymbol{x}) = (0.5x_1+0.1x_6; (0.5+d)x_2; x_3; 0.1x_1+0.4x_4; 0.2x_1+x_5; x_6; 0.1x_1+x_7)$, $X_0 = \{\boldsymbol{x} \mid \sum_{i=1}^{7} x_i^2 - 1 \geq 0\}$, $X_1 = \{\boldsymbol{x} \mid x_1+x_2+x_3-x_4-x_5-x_6-x_7 \geq 0\}$, $X_2 = \{(x, y) \mid x_1+x_2+x_3-x_4-x_5-x_6-x_7 < 0\}$ and $D = \{d \mid d^2 - 0.01 \leq 0\}$. From the computation times listed Table 1, we conclude that although the computation time increases with the number of variables increasing, our method may deal with problems with many variables, especially for the cases that the robust nontermination set formed by a polynomial of low degree fulfills certain needs in real applications. Note that numerical simulation techniques suffers from the curse of dimensionality and thus can not apply to this example since this example has seven variables, we just illustrate the results computed by our method based on (15) in Fig. 5.

5   Related Work

Methods for proving nontermination of programs have recently been studied actively. [16] uses a characterization of nontermination by recurrence sets of states that is visited infinitely often along the path. A recurrence set exists iff a program is non-terminating. To find recurrence sets they provide a method based on constraint solving. Their method is only applicable to programs with linear integer arithmetic and does not support non-determinism and is implemented in the tool TNT. [7] proposes a method combining closed recurrence sets with counterexample-guided underapproximation for disproving termination. This method, implemented in the tool T2, relies heavily on suitable safety provers for the class of programs of interest, thus rendering an application of
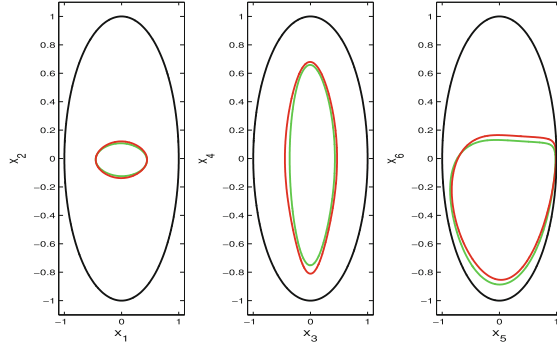
Fig. 5. Computed robust nontermination sets for Example 4. (Black, Red and Green curves – the boundaries of X_0 and the cross-sections (from left to right: $x_3 = x_4 = x_5 = x_6 = x_7 = 0$, $x_1 = x_2 = x_5 = x_6 = x_7 = 0$ and $x_1 = x_2 = x_3 = x_4 = x_7 = 0$) of the computed robust nontermination sets $\mathcal{R}_0$ when $d_u = 5$ and $d_u = 4$, respectively.) (Color figure online)

their method to nonlinear programs difficult. Further, [9] introduces live abstractions combing with closed recurrence sets to disprove termination. However, this method, implemented in the tool ANANT, is only suitable for disproving nontermination in the form of lasso for programs of finite control-flow graphs.

There are also some approaches exploiting theorem-proving techniques to prove nontermination, e.g., [53] presents a method for disproving nontermination of Java programs based on theorem proving and generation of invariants. This method is implemented in INVEL, which is restricted to deterministic programs with unbounded integers and single loops. APROVE [14] uses SMT solving to prove nontermination of Java programs [6]. The application of this method requires either singleton recurrence sets or loop conditions being recurrence sets in the programs of interest. [23] disproves termination based on MaxSMT-based invariant generation, which is implemented in the tool CPPINV. This method is limited to linear arithmetic as well.

Besides, TREX [17] integrates existing non-termination proving approaches to develop compositional analysis algorithms for detecting non-termination in multithreaded programs. Different from the method in TREX targeting sequential code, [1] presents a nontermination proving technique for multi-threaded programs via a reduction to nontermination reasoning for sequential programs. [27] investigates the termination problems of multi-path polynomial programs with equational loop guards and discovering nonterminating inputs for such programs. It shows that the set of all strong non-terminating inputs and weak non-terminating inputs both correspond to the real varieties of certain polynomial ideals. Recently, [22] proposes a method combining higher-order model checking with predicate abstraction and CEGAR for disproving nontermination of higher-order functional programs. This method reduces the problem of

disproving non-termination to the problem of checking a certain branching property of an abstract program, which can be solved by higher-order model checking.

Please refer to [10, 55] for detailed surveys on termination and nontermination analysis of programs.

As opposed to above works without considering robust nontermination, by taking disturbances such as round-off errors in performing numerical implementation of computer programs into account, this paper propose a systematic approach for proving robust nontermination of a class of computer programs, which are composed of a single loop with a possibly complicated switch-case type loop body and encountered often in current embedded systems. The problem of robust conditional nontermination is reduced to a problem of solving a single equation derived via dynamic programming principle, and semi-definite programs could be employed to solve such optimal control problem efficiently in some situations.

The underlying idea in this work is in sprit analogous to that in [40], which is pioneer in proposing a systematic framework to conduct verification of numerical software based on Lyapunov invariance in control theroy. Our method for conducting (robust) verification of numerical software falls within the framework proposed in [40]. The primary contribution of our work is that we systematically investigate a class of computer programs and reduce the nontermination problem for such computer programs to a mathematical equation, thus resulting in an efficient nontermination verification method, as indicated in Introduction Sect. 1.

6 Conclusion and Future Work

In this paper we presented a system-theoretic framework to numerical software analysis and considered the problem of conditional robust non-termination analysis for a class of computer programs composed of a single loop with a possibly complicated switch-case type loop body, which is encountered often in real-time embedded systems. The maximal robust nontermination set of initial configurations in our method was characterized by a solution to a mathematical equation. Although it is non-trivial to solve gained equation, in the case of polynomial assignments in the loop body and basic semi-algebraic sets in Program 1, the equation could be relaxed as a semi-definite program, which falls within the convex programming framework and can be solved efficiently via interior point methods. Finally, we have reported experiments with encouraging results to demonstrate the merits of our method.

However, there are a lot of works remaining to be done. For instance, the semi-definite programming solver is implemented with floating point computations, we have no absolute guarantee on the results it provides. In future work, we need a sound and efficient verification procedure such as that presented in [26, 35, 41, 46] that is able to check the result from the solver and help us decide whether the result is qualitatively correct. Besides, the presented work can be extended in several directions, these include robust nontermination analysis for computer programs with nested loops and robust invariant generations with or without constraints [20, 28, 42].

References

1. Atig, M.F., Bouajjani, A., Emmi, M., Lal, A.: Detecting fair non-termination in multithreaded programs. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 210–226. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_19
2. Blanchini, F., Miani, S.: Set-Theoretic Methods in Control. Springer, Boston (2008). https://doi.org/10.1007/978-0-8176-4606-6
3. Borralleras, C., Brockschmidt, M., Larraz, D., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving termination through conditional termination. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 99–117. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_6
4. Bourbaki, N.: General Topology: Chapters 1–4, vol. 18. Springer, Heidelberg (2013)
5. Boyd, S., El Ghaoui, L., Feron, E., Balakrishnan, V.: Linear Matrix Inequalities in System and Control Theory. SIAM, Philadelphia (1994)
6. Brockschmidt, M., Ströder, T., Otto, C., Giesl, J.: Automated detection of non-termination and NullPointerExceptions for Java Bytecode. In: Beckert, B., Damiani, F., Gurov, D. (eds.) FoVeOOS 2011. LNCS, vol. 7421, pp. 123–141. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31762-0_9
7. Chen, H.-Y., Cook, B., Fuhs, C., Nimkar, K., O'Hearn, P.: Proving nontermination via safety. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 156–171. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_11
8. Chen, Y.-F., Hong, C.-D., Wang, B.-Y., Zhang, L.: Counterexample-guided polynomial loop invariant generation by lagrange interpolation. In: Kroening, D., Pǎsǎreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 658–674. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_44
9. Cook, B., Fuhs, C., Nimkar, K., O'Hearn, P.: Disproving termination with over-approximation. In: Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design, pp. 67–74. FMCAD Inc. (2014)
10. Cook, B., Podelski, A., Rybalchenko, A.: Proving program termination. Commun. ACM **54**(5), 88–98 (2011)
11. Cousot, P., Cousot, R.: A gentle introduction to formal verification of computer systems by abstract interpretation (2010)
12. Fedkiw, S.O.R., Osher, S.: Level set methods and dynamic implicit surfaces. Surfaces **44**, 77 (2002)
13. Floyd, R.W.: Assigning meanings to programs. Math. Aspects Comput. Sci. **19**(19–32), 1 (1967)
14. Giesl, J., et al.: Proving termination of programs automatically with AProVE. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS (LNAI), vol. 8562, pp. 184–191. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08587-6_13
15. Giesl, P., Hafstein, S.: Review on computational methods for Lyapunov functions. Discrete Contin. Dyn. Syst.-Ser. B **20**(8), 2291–2331 (2015)
16. Gupta, A., Henzinger, T.A., Majumdar, R., Rybalchenko, A., Xu, R.-G.: Proving non-termination. ACM Sigplan Not. **43**(1), 147–158 (2008)
17. Harris, W.R., Lal, A., Nori, A.V., Rajamani, S.K.: Alternation for termination. In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 304–319. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15769-1_19

18. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (1969)
19. Jarvis-Wloszek, Z.W.: Lyapunov based analysis and controller synthesis for polynomial systems using sum-of-squares optimization. Ph.D. thesis, University of California, Berkeley (2003)
20. Kapur, D.: Automatically generating loop invariants using quantifier elimination. In: Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2006)
21. Kouramas, K.I., Rakovic, S.V., Kerrigan, E.C., Allwright, J., Mayne, D.Q.: On the minimal robust positively invariant set for linear difference inclusions. In: 44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference, CDC-ECC 2005, pp. 2296–2301. IEEE (2005)
22. Kuwahara, T., Sato, R., Unno, H., Kobayashi, N.: Predicate abstraction and CEGAR for disproving termination of higher-order functional programs. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9207, pp. 287–303. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21668-3_17
23. Larraz, D., Nimkar, K., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving non-termination using max-SMT. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 779–796. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_52
24. Lasserre, J.B.: Tractable approximations of sets defined with quantifiers. Math. Programm. **151**(2), 507–527 (2015)
25. Li, Y.: Witness to non-termination of linear programs. Theor. Comput. Sci. **681**, 75–100 (2017)
26. Lin, W., Wu, M., Yang, Z., Zeng, Z.: Exact safety verification of hybrid systems using sums-of-squares representation. Sci. China Inf. Sci. **57**(5), 1–13 (2014)
27. Liu, J., Xu, M., Zhan, N., Zhao, H.: Discovering non-terminating inputs for multipath polynomial programs. J. Syst. Sci. Complex. **27**(6), 1286–1304 (2014)
28. Liu, J., Zhan, N., Zhao, H.: Computing semi-algebraic invariants for polynomial dynamical systems. In: Proceedings of the Ninth ACM International Conference on Embedded Software, pp. 97–106. ACM (2011)
29. Lofberg, J.: YALMIP: a toolbox for modeling and optimization in MATLAB. In: 2004 IEEE International Symposium on Computer Aided Control Systems Design, pp. 284–289. IEEE (2004)
30. Luk, C.K., Chesi, G.: On the estimation of the domain of attraction for discrete-time switched and hybrid nonlinear systems. Int. J. Syst. Sci. **46**(15), 2781–2787 (2015)
31. Magron, V., Garoche, P.-L., Henrion, D., Thirioux, X.: Semidefinite approximations of reachable sets for discrete-time polynomial systems. arXiv preprint arXiv:1703.05085 (2017)
32. Mitchell, I.M., Bayen, A.M., Tomlin, C.J.: A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. IEEE Trans. Autom. Control **50**(7), 947–957 (2005)
33. Mosek, A.: The MOSEK optimization toolbox for MATLAB manual. Version 7.1 (Revision 28), p. 17 (2015)
34. Naur, P.: Proof of algorithms by general snapshots. BIT Numer. Math. **6**(4), 310–316 (1966)
35. Platzer, A., Quesel, J.-D., Rümmer, P.: Real world verification. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 485–501. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02959-2_35

36. Prajna, S., Jadbabaie, A.: Safety verification of hybrid systems using barrier certificates. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 477–492. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24743-2_32
37. Prajna, S., Jadbabaie, A., Pappas, G.J.: A framework for worst-case and stochastic safety verification using barrier certificates. IEEE Transa. Autom. Control **52**(8), 1415–1428 (2007)
38. Rakovic, S.V., Kerrigan, E.C., Kouramas, K.I., Mayne, D.Q.: Invariant approximations of the minimal robust positively invariant set. IEEE Trans. Autom. Control **50**(3), 406–410 (2005)
39. Rebiha, R., Matringe, N., Moura, A.V.: Generating asymptotically non-terminating initial values for linear programs. arXiv preprint arXiv:1407.4556 (2014)
40. Roozbehani, M., Megretski, A., Feron, E.: Optimization of Lyapunov invariants in verification of software systems. IEEE Trans. Autom. Control **58**(3), 696–711 (2013)
41. Roux, P., Voronin, Y.-L., Sankaranarayanan, S.: Validating numerical semidefinite programming solvers for polynomial invariants. In: Rival, X. (ed.) SAS 2016. LNCS, vol. 9837, pp. 424–446. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53413-7_21
42. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Non-linear loop invariant generation using gröbner bases. ACM SIGPLAN Not. **39**(1), 318–329 (2004)
43. Sassi, M.A.B., Girard, A.: Controller synthesis for robust invariance of polynomial dynamical systems using linear programming. Syst. Control Lett. **61**(4), 506–512 (2012)
44. Sassi, M.A.B., Girard, A., Sankaranarayanan, S.: Iterative computation of polyhedral invariants sets for polynomial dynamical systems. In: 2014 IEEE 53rd Annual Conference on Decision and Control (CDC), pp. 6348–6353. IEEE (2014)
45. Schaich, R.M., Cannon, M.: Robust positively invariant sets for state dependent and scaled disturbances. In: 2015 IEEE 54th Annual Conference on Decision and Control (CDC), pp. 7560–7565. IEEE (2015)
46. Sturm, T., Tiwari, A.: Verification and synthesis using real quantifier elimination. In: Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation, pp. 329–336. ACM (2011)
47. Tahir, F., Jaimoukha, I.M.: Robust positively invariant sets for linear systems subject to model-uncertainty and disturbances. IFAC Proc. Vol. **45**(17), 213–217 (2012)
48. Tiwari, A.: Termination of linear programs. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 70–82. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27813-9_6
49. Topcu, U., Packard, A., Seiler, P.: Local stability analysis using simulations and sum-of-squares programming. Automatica **44**(10), 2669–2675 (2008)
50. Topcu, U., Packard, A.K., Seiler, P., Balas, G.J.: Robust region-of-attraction estimation. IEEE Trans. Autom. Control **55**(1), 137–142 (2010)
51. Trodden, P.: A one-step approach to computing a polytopic robust positively invariant set. IEEE Trans. Autom. Control **61**(12), 4100–4105 (2016)

52. Vandenberghe, L., Boyd, S.: Semidefinite programming. SIAM Rev. **38**(1), 49–95 (1996)
53. Velroyen, H., Rümmer, P.: Non-termination checking for imperative programs. In: Beckert, B., Hähnle, R. (eds.) TAP 2008. LNCS, vol. 4966, pp. 154–170. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79124-9_11
54. Xia, B., Yang, L., Zhan, N., Zhang, Z.: Symbolic decision procedure for termination of linear programs. Formal Aspects Comput. **23**(2), 171–190 (2011)
55. Yang, L., Zhou, C., Zhan, N., Xia, B.: Recent advances in program verification through computer algebra. Front. Comput. Sci. China **4**(1), 1–16 (2010)