Polynomial Reachability Witnesses via Stellensätze

Ali Asadi, Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, Mohammad Mahdavi

Polynomial Reachability Witnesses via Stellensätze

Ali Asadi*
Sharif University of Technology
Tehran, Iran
aasadi@ce.sharif.edu

Krishnendu Chatterjee
IST Austria
Klosterneuburg, Austria
krishnendu.chatterjee@ist.ac.at

Hongfei Fu
Shanghai Jiao Tong University
Shanghai, China
fuhf@cs.sjtu.edu.cn

Amir Kafshdar Goharshady†
The Hong Kong University of Science and Technology
Hong Kong, China
goharshady@cse.ust.hk

Mohammad Mahdavi
Sharif University of Technology
Tehran, Iran
mimmahdavi@ce.sharif.edu

Abstract

We consider the fundamental problem of reachability analysis over imperative programs with real variables. Previous works that tackle reachability are either unable to handle programs consisting of general loops (e.g. symbolic execution), or lack completeness guarantees (e.g. abstract interpretation), or are not automated (e.g. incorrectness logic). In contrast, we propose a novel approach for reachability analysis that can handle general and complex loops, is complete, and can be entirely automated for a wide family of programs. Through the notion of *Inductive Reachability Witnesses* (IRWs), our approach extends ideas from both invariant generation and termination to reachability analysis.

We first show that our IRW-based approach is sound and complete for reachability analysis of imperative programs. Then, we focus on linear and polynomial programs and develop automated methods for synthesizing linear and polynomial IRWs. In the linear case, we follow the well-known approaches using Farkas' Lemma. Our main contribution is in the polynomial case, where we present a push-button semi-complete algorithm. We achieve this using a novel combination of classical theorems in real algebraic geometry, such as Putinar's Positivstellensatz and Hilbert's Strong Nullstellensatz. Finally, our experimental results show we can prove complex reachability objectives over various benchmarks that were beyond the reach of previous methods.

CCS Concepts: • **Theory of computation → Logic and verification**; **Automated reasoning**; • **Software and its engineering → Formal software verification**.

Keywords: Reachability, Inductive Reasoning, Stellensätze

1 Introduction

Reachability. *Reachability analysis* is a basic and fundamental problem in computer science, starting from the halting problem of Turing machines. It is a core problem in program verification, constitutes the most basic liveness property, and has been widely studied in program analysis and model checking [39, 57, 70, 81, 87]. The target states considered in reachablity analysis can be either *desirable* so that reachability to these states should be guaranteed, or *undesirable* so that the goal is to find an execution path leading to an unwanted behavior, hence proving incorrectness of the system. Reachability to desirable states encodes the most basic type of liveness property, but reachability to undesirable states is also ubiquitous in verification problems and useful when one needs to identify realistic bugs in software implementations (see e.g. [86]). Indeed, in real-world software development, most bugs are identified by finding an execution path that leads to a specific error [52, 62, 80]. This idea led to developments such as incorrectness logic [86].

Previous Works on Formal Models. A large body of research on reachability analysis is conducted over formal models [39], such as finite-state systems [12, Chapter 3–6], Petri nets [8, 46, 47, 83] and timed automata [6]. For these models, precise decidability and complexity results are attained. Although these models serve as an important abstraction mechanism for realistic systems, the techniques for reachability analysis over them cannot be applied directly to imperative programs, because the values taken by variables in a program typically come from an infinite, even uncountable, domain and the underlying program structure might be irregular. In many cases a given program cannot be translated into any of the formal models above.

Reachability in Software Model Checkers. Many model checkers rely heavily on reachability analysis [16, 24, 25, 71]. Notably, the BLAST project [24] describes itself as "a verification tool for the C language that solves the reachability problem". Even when considering safety properties, all approaches based on Counterexample-Guided Abstraction Refinement (CEGAR) [7, 14, 38, 65, 66], including SLAM [15, 16] and BLAST [24], need to constantly perform reachability analyses to obtain their counterexamples. These model checkers rely on predicate abstraction refinement and, assuming a termination requirement, guarantee completeness when the variables have finite domains [68].

*Authors are listed in alphabetical order.
†Corresponding Author. Part of the results are included in [63].

Previous Works on (Imperative) Programs. When considering imperative programs, the reachability problem, and in particular the special case of termination analysis, has been widely studied over the past decades. Previous works include symbolic execution [29, 30, 74], termination analysis [57], abstract interpretation [44] and recent results on incorrectness logic [50, 86].

- *Symbolic execution* runs program code statically in a symbolic fashion, and is thus effective for programs without general unbounded loops. For programs with loops, symbolic execution can only unfold the loop up to a bounded depth, and hence cannot handle general loops with an unbounded number of iterations. This is also applicable to other approaches that rely on loop unrolling, such as [3].
- *Termination analysis* is a special kind of reachability which is usually guaranteed by well-foundedness reasoning such as (lexicographic) ranking functions [5, 17, 18, 26, 27, 41, 42, 88, 89]. It does not consider target states defined through numerical constraints over program variables.
- *Abstract interpretation* is mainly used to generate over-approximations of reachable states (i.e. certain states *may* be reached), but there are also several abstraction-based approaches that compute under-approximations [3, 4, 61, 91, 93, 96]. However, they cannot provide guarantees of completeness except in specific special cases [60].
- Finally, *incorrectness logic* [86] is a sound and complete logic that is similar to Hoare logic but performs under-approximation for reachable program states. A disadvantage of incorrectness logic, much like Hoare logic, is that it requires a considerable amount of manual effort for writing assertions, and cannot be directly automated.

Previous Works on Invariants. An *invariant* is, in a sense, a dual notion of reachability, and invariant generation is also prominent in the PL literature. Informally, an invariant is an over-approximation of the set of reachable states that can be used to prove safety properties over programs. Invariant generation is widely-studied in program analysis and verification. Many approaches are considered, e.g. abstract interpretation [10, 98], constraint solving [36, 40, 94, 95] and abductive inference [51].

Our Focus. We consider reachability analysis over imperative programs and study the problem of automatically verifying that a set of target states can be reached in program execution. While invariants provide an over-approximation of the set of reachable states, we consider their natural dual, i.e. under-approximations of the set of states that can reach a target. We consider programs with non-determinism and distinguish between *existential* and *universal* reachability. *Existential* reachability is the more classical notion and requires that target states are reachable under *some* resolution of the nondeterminism. *Universal* reachability requires the program to reach the target states no matter how the nondeterminism is resolved. We focus on existential reachability, but our results generalize to the universal case, too.

Our Approach. Our methods are based on classical theorems (stellensätze) in polyhedral and real algebraic geometry. We first extend ideas from both ranking functions and inductive invariant generation to cover the reachability problem. Informally, we use techniques from inductive invariant generation to capture a subset $\mathbf{T}^\diamond$ of program states from which the execution steps of the program will either reach our target states or stay in $\mathbf{T}^\diamond$ itself. Simultaneously, we use arguments similar to ranking functions to ensure that every state in $\mathbf{T}^\diamond$ can reach a target state in finitely many steps. While using an invariant and a ranking function to prove termination is a classical approach, the key distinction is that our set $\mathbf{T}^\diamond$ is an *under-approximation* of the set of states that can eventually reach a target state, whereas invariants are *over-approximations* of reachable states. Moreover, our inductive sets are closed *existentially* (for each state in the inductive set, some successor should be in the same set), whereas inductive invariants are closed *universally* (all successors should be in the set). A similar approach based on *danger invariants* has been proposed in [48]. A detailed comparison will be presented in the sequel.

Our Contributions. We propose a novel approach for reachability analysis over programs with real variables*. Our contributions are:

- We propose the notion of Inductive Reachability Witnesses (IRWs) for existential reachability, which consists of a set $\mathbf{T}^\diamond$ of program states and a ranking function f over $\mathbf{T}^\diamond$. The state set $\mathbf{T}^\diamond$ satisfies certain invariant-like conditions. The ranking function f serves as a proof that every state in $\mathbf{T}^\diamond$ can indeed reach a target. We also propose the notion of Universal Inductive Reachability Witnesses (UIRWs), the counterpart of IRWs for the universal case.
- From a theoretical point-of-view, we show that IRWs and UIRWs are sound and complete for proving existential and universal reachability, respectively.
- Similar to several previous template-based works [36, 40, 94, 95], we use Farkas' Lemma, Putinar's Positivstellensatz, and Handelman's Theorem for automatically synthesizing linear and polynomial IRWs/UIRWs. However, we face new challenges regarding satisfiability in the polynomial case and address them with novel methods based on Hilbert's Strong Nullstellensatz. This is the main technical novelty of this work. To the best of our knowledge, such methods (Section 3.2 and Theorem 8) are a novel contribution to constraint-based analysis of polynomial programs, and have no parallels even in the context of invariant generation or termination analysis. Moreover,

*Note that our approaches cannot be directly applied to floating-point numbers, as abstracting them to reals is unsound.

our *synthesis algorithms* are *complete* in the linear case and *semi-complete* in the polynomial case.

Comparison with Danger Invariants. Structures similar to IRWs, considering inductive sets that are used as under-approximations for reachability, have been studied in various contexts, including in [13, 21, 37, 43, 101]. The most similar previous work is [48] in which *danger invariants* are introduced. Like our notion of IRW, a (partial) danger invariant consists of an inductive set and a ranking function. However, we differ from [48] in a number of significant ways:

- *Problem:* We can handle both universal and existential reachability, whereas [48] is limited to the existential case.
- *Algorithm:* Our synthesis algorithms are based on a novel combination of null- and positivstellensätze, while [48]'s approach is based on symbolic execution, evolutionary mutations, and second-order SAT solving. Our stellensätze-based approach is the central contribution of this work.
- *Completeness of the Synthesis Algorithms:* We consider linear and polynomial programs and witnesses. Our push-button *synthesis algorithms* are complete for the linear (Theorem 4) and semi-complete for the polynomial case (Theorem 10), while [48] does not provide completeness.
- *Conceptual:* [48] attempts to synthesize an explicit scheduler in order to handle non-determinism. In contrast, we directly encode the non-determinism in the inputs to our stellensätze and leave the scheduler implicit.
- *Practical:* Our experimental results in Section 4 demonstrate that our approach can handle instances that are beyond the reach of [48]. There are also programs which are not expressible in our framework, due to being non-polynomial, but can be handled by [48].

Deep Bugs. A major challenge in reachability analysis is posed by *deep bugs*. If undesirable states are only reachable after a very long execution, then finite model checking or symbolic execution will have a hard time identifying the bug. Works that address this problem include [48, 58, 77, 79]. Our runtime does not depend on the depth of the bug and we can identify various deep bugs with ease. See Section 4.

Comparison with Approaches for Deep Bugs. Although the main focus of our approach is providing completeness results in the linear and polynomial cases, which no previous method can provide, our approach is also very effective in proving the existence of deep bugs, i.e. proving reachability properties that can only be attained through long execution paths. As such, we now compare with several other approaches for deep bugs:

- The work of [77] focuses mostly on deep bugs in programs that manipulate arrays. It considers under-approximating the behavior of loops using the notion of "auxiliary paths". This is similar to loop acceleration/summarization but can handle more general loops, given that it under-approximates the behavior of the loop rather than trying to match it exactly. This method is sound and applicable to a wide family of programs. However, it does not provide completeness guarantees. In contrast, our approach is sound and complete for linear and polynomial programs.
- Another work in this category is [58]. This work focuses on summarizing and under-approximating the behavior of recursive functions and identifying bugs that can only be reached using deep recursion (a long stack). Another major difference between our work and [58] is that [58] focuses on integer programs, whereas our transition systems are defined with real variables. Finally, we also provide completeness guarantees.
- Finally, [79] explores compositional approaches to proving reachability (safety refutation) and provides three algorithms with various levels of completeness. However, these algorithms are only applicable to loop-free programs, whereas we can handle (nested) loops.

2 Inductive Reachability Witnesses

We now provide the basic definitions needed for reachability analysis and introduce the concept of Inductive Reachability Witnesses (IRWs/UIRWs). In the sequel, we use transition systems with real variables to model programs.

Valuations. Let $\mathbf{V}$ be a finite set of *variables*. A *valuation* over $\mathbf{V}$ is a function $\nu : \mathbf{V} \to \mathbb{R}$. We denote the set of all valuations over $\mathbf{V}$ by $\mathbb{R}^{\mathbf{V}}$.

Transition Systems. A *transition system* (or simply *system*) is a tuple $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, in which $\mathbf{V}$ is a finite set of variables, $\mathbf{L}$ is a finite set of *locations*, $\ell_0 \in \mathbf{L}$ is the *initial* or *starting* location, I is an assertion over $\mathbf{V}$ which defines the set of possible initial valuations, and Θ is a finite set of *transitions*. Each transition $\theta \in \Theta$ is of the form $\theta = (\ell, \ell', \varphi, \mu)$ where $\ell, \ell' \in \mathbf{L}$ are the *pre* and *post* locations, φ is an assertion over $\mathbf{V}$ that serves as the *transition condition*, and $\mu : \mathbb{R}^{\mathbf{V}} \to \mathbb{R}^{\mathbf{V}}$ is an *update function*. For brevity, in the sequel, we assume that we have fixed a system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$ which is under study. For a location $\ell \in \mathbf{L}$, we write Θ_ℓ to denote the set of transitions out of ℓ. We say that a system is β-branching if $|\Theta_\ell| \leq \beta$ for every location ℓ.

Example 1. *Consider the program in Figure 1 (top), in which $\square$ denotes non-deterministic choice between transitioning to b or c. The transition system in Figure 1 (bottom) represents this program. Note that we have $\ell_0 = a$ and assume the initial valuations satisfy $x, y, z \geq 0$.*

States. A state in S is a pair $\sigma = (\ell, \nu) \in \mathbf{L} \times \mathbb{R}^{\mathbf{V}}$, consisting of a location and a valuation. We denote the set of states by Σ. A subset $\Sigma' \subseteq \Sigma$ is called bounded if the set of valuations that appear in the elements of Σ' is bounded.

Successors. A state $\sigma' = (\ell', \nu')$ is called a *successor* of a state $\sigma = (\ell, \nu)$ if there exists a transition $\theta = (\ell, \ell', \varphi, \mu) \in \Theta$ such that $\nu \models \varphi$ and $\nu' = \mu(\nu)$. *For theoretical elegance, we assume that every state has at least one successor*, i.e. we assume that there is a transition from the final state to itself.

$I : x \geq 0 \ \wedge \ y \geq 0 \ \wedge \ z \geq 0$
$a :$ **while** $x \geq y :$
$b :$ $\quad \square\ (x, y) := (x + 1, y + 2)$
$c :$ $\quad \square\ (x, y, z) := (x + z, y + z, z - 1)$
$d :$

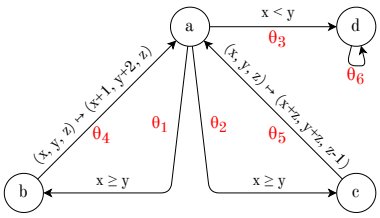

Figure 1. A Simple Program (top) and its Representation as a Transition System (bottom)

Example 2. *In Figure 1,* $(b, 1, 1, 2)$*, i.e. the state at location* b *for which the values of* x *and* y *are* 1 *and the value of* z *is* 2*, is a successor of* $(a, 1, 1, 2)$ *through* θ_1*. Similarly,* $(a, 2, 3, 2)$ *is a successor of* $(b, 1, 1, 2)$ *through* θ_4*.*

Runs. A *run* of the system S is an *infinite* sequence $\mathsf{r} = \{\sigma_i, \theta_i\}_{i=0}^{\infty} = \{(\ell_i, \nu_i), \theta_i\}_{i=0}^{\infty}$, where each $\sigma_i \in \Sigma$ is a state consisting of $\ell_i \in \mathbf{L}$ and $\nu_i \in \mathbb{R}^{\mathbf{V}}$, and each $\theta_i = (\ell_i, \ell_{i+1}, \varphi_i, \mu_i) \in \Theta$ is a transition from ℓ_i to ℓ_{i+1}, such that:

- r starts in the initial location ℓ_0;
- $\nu_0 \models I$, i.e. the initial valuation satisfies the assertion I;
- For every i, we have $\nu_i \models \varphi_i$ and $\nu_{i+1} = \mu_i(\nu_i)$, i.e. σ_{i+1} is a successor of σ_i through θ_i.

Semi-runs. A *semi-run* is defined similarly to a run, except that it does not have to start at ℓ_0 or satisfy I. A *path* of length n is a finite prefix $\pi = \sigma_0, \theta_0, \ldots, \sigma_{n-1}, \theta_{n-1}, \sigma_n$ of a run. A path must always end at a state. Similarly, a *semi-path* is a finite prefix of a semi-run that ends at a state.

Non-determinism. The system S is called *deterministic* if there is exactly one possible transition at every state. Formally, S is deterministic if for every $\sigma = (\ell, \nu) \in \Sigma$, there exists exactly one $\theta \in \Theta$ such that $\theta = (\ell, \ell', \varphi, \mu)$ and $\nu \models \varphi$. Otherwise, S is *non-deterministic*.

Existential Reachability. A set $\mathbf{T} \subseteq \Sigma$ is called *existentially reachable* or simply *reachable* if there exists an integer n and a run $\mathsf{r} = \{\sigma_i, \theta_i\}_{i=0}^{\infty}$ such that $\sigma_n \in \mathbf{T}$. In other words, $\mathbf{T}$ is reachable if there exists a run that visits $\mathbf{T}$.

Universal Reachability. A set $\mathbf{T} \subseteq \Sigma$ is called *universally reachable* if there exists a valuation $\nu_0 \in \mathbb{R}^{\mathbf{V}}$ and an integer n, such that (i) $\nu_0 \models I$, and (ii) every run $\mathsf{r} = \{(\ell_i, \nu_i), \theta_i\}_{i=0}^{\infty}$ visits $\mathbf{T}$ in its first n steps. In other words, for each such r, there exists an index $i \leq n$ such that $(\ell_i, \nu_i) \in \mathbf{T}$.

Intuitively, this requires that we can fix an initial valuation such that no matter how the non-determinism is resolved, the execution is forced to visit $\mathbf{T}$ after at most n steps. Note that our notion of universal reachability is bounded by n and different from having $\Diamond\mathbf{T}$ in every execution path.

Motivation for Universal Reachability. Universal reachability is useful for modeling angelic non-determinism. Consider that $\mathbf{T}$ is a set of undesirable (bug) states and the question is whether for all initial valuations, there exists an execution that avoids the bug. For example, consider a cyber-physical system modeled as a transition system and a controller who can resolve non-determinism. Deciding (the negation of) universal reachability corresponds to whether for all choices of initial state, a controller can avoid the bug.

Example 3. *Consider the system in Figure 1 (bottom), and let* $\mathbf{T} = \{(d, \nu) \mid \nu \in \mathbb{R}^{\mathbf{V}}\}$*. Reaching* $\mathbf{T}$ *is equivalent to the termination of the program in Figure 1 (top).* $\mathbf{T}$ *is existentially reachable, i.e. there are runs of the system that reach label* d*, for example the following:*

$$(a, 0, 0, 0) \xrightarrow{\theta_1} (b, 0, 0, 0) \xrightarrow{\theta_4} (a, 1, 2, 0) \xrightarrow{\theta_3} (d, 1, 2, 0) \rightarrow \ldots.$$

It is also universally reachable, because every execution starting from $(a, 1, 2, 3)$ *will reach* d *in a single step. As another example, consider the target set* $\mathbf{T}' = \{(d, \nu) \mid \nu(x) < 0\}$*. This corresponds to reaching* d *(ending the program) with a negative value for* x*. This time, the set* $\mathbf{T}'$ *is existentially reachable, for example through the following run:*

$$(a, 0, 0, 0) \xrightarrow{\theta_2} (c, 0, 0, 0) \xrightarrow{\theta_5} (a, 0, 0, -1) \xrightarrow{\theta_2} (c, 0, 0, -1) \xrightarrow{\theta_5} (a, -1, -1, -2) \xrightarrow{\theta_2} (c, -1, -1, -2) \xrightarrow{\theta_5} (a, -3, -3, -3) \xrightarrow{\theta_1} (b, -3, -3, -3) \xrightarrow{\theta_4} (a, -2, -1, -3) \xrightarrow{\theta_3} (d, -2, -1, -3) \rightarrow \ldots,$$

but it is not universally reachable. To see this, note that if an initial value satisfies $x < y$*, then it does not enter the while loop at all, and hence when it reaches* d *it satisfies* $x \geq 0$ *(the initial condition). On the other hand, if an initial value satisfies* $x \geq y$*, there is a run that always chooses the transition* θ_2 *when at* a*, and hence never reaches* $\mathbf{T}'$*.*

We now look into proof concepts for reachability.

$\mathbf{T}$-inductive Sets. Given a set $\mathbf{T} \subseteq \Sigma$ of target states, a set $\mathbf{T}^{\Diamond} \subseteq \Sigma$ is called $\mathbf{T}$-*inductive* if for every $\sigma \in \mathbf{T}^{\Diamond} \setminus \mathbf{T}$, *there exists* a successor σ' of σ such that $\sigma' \in \mathbf{T}^{\Diamond}$.

Intuitively, if $\mathbf{T}^{\Diamond}$ is $\mathbf{T}$-inductive, then if we start the execution of the program from a state in $\mathbf{T}^{\Diamond}$, there exists a way for resolving the non-determinism so that we either reach $\mathbf{T}$ or can inductively prove that we will never leave $\mathbf{T}^{\Diamond}$.

Example 4. *Consider the system in Figure 1 and let* $\mathbf{T} = \{(d, \nu) \mid \nu(x) < 0\}$*, i.e. the target is reaching* d *with* x *having a negative value. Let* $\mathbf{T}^{\Diamond} := \{(\ell, \nu) \mid \ell \in \mathbf{L}, \nu \in \mathbb{R}^{\mathbf{V}}, \nu \models A_\ell\}$ *be the set of states satisfying the following assertions:*

ℓ	A_ℓ
a	$x, y, z \leq 0 \ \wedge \ (x - y) \cdot (x - y + 1) = 0$
b	$x \leq -2 \ \wedge \ y, z \leq 0 \ \wedge \ x = y$
c	$x, y, z \leq 0 \ \wedge \ x = y$
d	$x < 0$

Then, we can verify that $\mathbf{T}^{\Diamond}$ *is a* $\mathbf{T}$*-inductive set. Concretely, consider a state* $(a, \nu_a) \in \mathbf{T}^{\Diamond}$*. In other words,* $\nu_a \models A_a$*. In such a state, we have* $(x - y) \cdot (x - y + 1) = 0$*. Therefore, either*

$\nu_a(x) = \nu_a(y)$ *or* $\nu_a(x) = \nu_a(y) - 1$. *In the former case, we can take transition* θ_2, *and it is easy to verify that the new state satisfies* A_c, *hence there is a successor that is also in* $\mathbf{T}^\diamond$. *In the latter case, we can take* θ_3 *and reach* d *with a valuation that satisfies* $x < 0$, *because* $\nu_a \models (y \le 0 \wedge x = y-1)$. *Similarly, if* $(b, \nu_b) \in \mathbf{T}^\diamond$, *we know that* $\nu_b \models (x \le -2 \wedge y, z \le 0 \wedge x = y)$. *Therefore, taking the transition* θ_4, *corresponding to the update* $(x, y) := (x{+}1, y{+}2)$, *leads to a state in* a *that satisfies* $(x, y, z \le 0 \wedge x = y-1)$. *Note that* $x = y-1 \Rightarrow (x-y)\cdot(x-y+1) = 0$, *therefore* A_a *is satisfied and we have a successor in* $\mathbf{T}^\diamond$. *It is easy to verify the same property at* c. *Finally, if we have a state* $(d, \nu_d) \in \mathbf{T}^\diamond$, *then it is also in* $\mathbf{T}$.

If we start at an initial state that satisfies A_a, *we can find a run of the system that either reaches* $\mathbf{T}$ *or stays inside* $\mathbf{T}^\diamond$. *However, this is not enough for reachability to* $\mathbf{T}$. *Such a run might stay inside* $\mathbf{T}^\diamond$ *forever without visiting* $\mathbf{T}$. *For example, we can keep taking the transition* θ_2 *when at* a, *and hence never reach* d. *To avoid this, we need a* $\mathbf{T}$*-ranking function.*

$\mathbf{T}$-*ranking Functions*. Given a $\mathbf{T}$-inductive set $\mathbf{T}^\diamond$, a function $f : \mathbf{T}^\diamond \to [0, \infty)$ is called a $\mathbf{T}$-*ranking function* with parameter $\epsilon > 0$, if for every $\sigma \in \mathbf{T}^\diamond \setminus \mathbf{T}$, *there exists* a successor $\sigma' \in \mathbf{T}^\diamond$ of σ, for which we have $f(\sigma') \le f(\sigma) - \epsilon$.

Inductive Reachability Witnesses (IRWs). Given a set $\mathbf{T}$ of target states in a system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, an *Inductive Reachability Witness* for $\mathbf{T}$ is a tuple $(\mathbf{T}^\diamond, f, \epsilon)$ such that:

- $\mathbf{T}^\diamond$ is a $\mathbf{T}$-inductive set;
- $\epsilon \in (0, \infty)$;
- $f : \mathbf{T}^\diamond \to [0, \infty)$ is $\mathbf{T}$-ranking with parameter ϵ;
- There exists $\nu \in \mathbb{R}^{\mathbf{V}}$ such that $(\ell_0, \nu) \in \mathbf{T}^\diamond$ and $\nu \models I$.

Informally, an IRW serves as a proof of existential reachability for a target set $\mathbf{T}$. The inductivity of $\mathbf{T}^\diamond$ ensures that starting from the initial state $(\ell_0, \nu) \in \mathbf{T}^\diamond$, we will never be forced to leave $\mathbf{T}^\diamond$ unless we reach $\mathbf{T}$, while the existence of the $\mathbf{T}$-ranking function f proves that we cannot avoid $\mathbf{T}$ forever. It is also noteworthy that the $\mathbf{T}$-inductive set $\mathbf{T}^\diamond$ is similar to an inductive invariant, but the main difference is that while an invariant is by definition a *superset* of all reachable states, a $\mathbf{T}$-inductive set $\mathbf{T}^\diamond$ is a *subset* of those states from which we can reach the target set $\mathbf{T}$. An IRW $(\mathbf{T}^\diamond, f, \epsilon)$ is called bounded if $\mathbf{T}^\diamond$ is bounded.

Example 5. *Consider the system in Figure 1, with the same target set as in Example 4, i.e.* $\mathbf{T} = \{(d, \nu) \mid \nu(x) < 0\}$. *Let* $\mathbf{T}^\diamond := \{(\ell, \nu) \mid \nu \models A_\ell\}$ *and* $f(\ell, \nu) := f_\ell(\nu)$ *be defined as:*

ℓ	A_ℓ	f_ℓ
a	$-10 \le x, y, z \le 0 \wedge \left(x = y - 1 \vee x = y = \frac{-z\cdot(z+1)}{2}\right)$	$100 + x - y + z$
b	$-10 \le x \le -2 \ \wedge\ z \le 0 \ \wedge\ x = y = \frac{-z\cdot(z+1)}{2}$	$99.5 + z$
c	$-2 \le x \le 0 \ \wedge\ z \le 0 \ \wedge\ x = y = \frac{-z\cdot(z+1)}{2}$	$99.5 + z$
d	$x \le -0.5$	0

Note that the A_ℓ*'s are more restrictive than in Example 4. We can verify that* $\mathbf{T}^\diamond$ *is a* $\mathbf{T}$*-inductive set in the same manner as in Example 4. We should also verify that* f *is a valid* $\mathbf{T}$*-ranking function. Whenever we take either transition* θ_1 *or* θ_2 *(from* a *to* b *or* c*), we are assured that* $x = y$, *hence the value of* f *goes from* $100 + z$ *to* $99.5 + z$ *and decreases by* 0.5. *Also, because in* A_a *we have* $-10 \le x, y, z \le 0$, *the value of* f *at* a *is at least* 80, *and hence transition* θ_3 *(from* a *to* d*) decreases* f *by more than* 0.5. *Now consider transition* θ_4 *(from* b *to* a*). This transition does not change the value of* z, *but makes it so that* $y = x + 1$. *So it changes the value of* f *from* $99.5 + z$ *to* $99 + z$. *Note that transition* θ_5 *(from* c *to* a*), decreases* z *by* 1 *while keeping* $x = y$. *Hence, it decreases* f *by* 0.5. *Also,* θ_6 *(the self-transition from* d *to* d*) is irrelevant in this case, because our* A_d *entails inclusion in* $\mathbf{T}$. *Finally,* $(a, 0, 0, 0)$ *satisfies both* I *and* A_a.

UIRWs. A Universal Inductive Reachability Witness (UIRW) is defined similarly, except that existential quantifiers in $\mathbf{T}^\diamond$ and f are replaced by universal quantifiers. See Appendix A.

Remark 1. *In* existential *IRWs (such as Example 5), the set* $\mathbf{T}^\diamond$ *is an under-approximation of the states from which there* exists *a way of resolving the non-determinism so that we eventually reach* $\mathbf{T}$. *Similarly, in UIRWs (Appendix A),* $\mathbf{T}^\diamond$ *under-approximates the set of states from which every execution of the program is forced to visit* $\mathbf{T}$. *Hence, our* $\mathbf{T}$*-inductive sets* $\mathbf{T}^\diamond$ *are essentially natural duals of inductive invariants [36, 40].*

We prove existential (universal) reachability by synthesizing an IRW (a UIRW). This is both sound and complete.

Theorem 1 (Soundness, Proof in Appendix B)**.** *Let* $\mathbf{T} \subseteq \Sigma$ *be a set of states in the system* S.

(i) If there exists an IRW $(\mathbf{T}^\diamond, f, \epsilon)$ *for* $\mathbf{T}$, *then* $\mathbf{T}$ *is existentially reachable.*

(ii) If there exists a UIRW $(\mathbf{T}^\diamond, f, \epsilon)$ *for* $\mathbf{T}$, *then* $\mathbf{T}$ *is universally reachable.*

Theorem 2 (Completeness, Proof in Appendix B)**.** *Let* $\mathbf{T} \subseteq \Sigma$ *be a set of states in the system* S.

(i) If $\mathbf{T}$ *is existentially reachable, then there exists an IRW* $(\mathbf{T}^\diamond, f, \epsilon)$ *for* $\mathbf{T}$.

(ii) If $\mathbf{T}$ *is universally reachable, then there exists a UIRW* $(\mathbf{T}^\diamond, f, \epsilon)$ *for* $\mathbf{T}$.

Undecidability. Synthesis of IRWs (UIRWs) is equivalent to proving existential (universal) reachability, which are undecidable due to Rice's theorem [92]. We consider the special case of linear or polynomial systems, with target sets that are defined by linear/polynomial inequalities, and focus on synthesizing linear/polynomial IRWs and UIRWs.

Polynomial Systems. A transition system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$ is called (d, k)*-polynomial* if

- I is a conjunction of at most k polynomial inequalities of degree at most d over $\mathbf{V}$, and
- for every $\theta = (\ell, \ell', \varphi, \mu) \in \Theta$, the transition condition φ is a conjunction of at most k polynomial inequalities of degree at most d over $\mathbf{V}$, and

- for every $\theta = (\ell, \ell', \varphi, \mu) \in \Theta$ and variable $v \in \mathbf{V}$, we have $\mu(v) \in \mathbb{R}[\mathbf{V}]$ and $\deg(\mu(v)) \leq d$, i.e. $\mu(v)$ is a polynomial of degree at most d over $\mathbf{V}$.

A $(1,k)-$polynomial system is also called *$k-$linear*.

Linear IRWs/UIRWs. An IRW/UIRW $(\mathbf{T}^\diamond, f, \epsilon)$ is called $k-$linear if for every location $\ell \in \mathbf{L}$:

- The set $\mathbf{T}^\diamond_\ell := \mathbf{T}^\diamond \cap (\{\ell\} \times \mathbb{R}^{\mathbf{V}})$ is a closed polyhedron which is an intersection of at most k half-spaces, i.e. there exists a set A_ℓ of at most k non-strict linear inequalities over $\mathbf{V}$ such that a valuation ν satisfies A_ℓ iff $(\ell, \nu) \in \mathbf{T}^\diamond$.
- The function $f_\ell : \textsc{Sat}(A_\ell) \to [0, \infty)$, defined as $f_\ell(\nu) = f(\ell, \nu)$, is a linear function over $\mathbf{V}$. Here, $\textsc{Sat}(A_\ell)$ is the set of all valuations that satisfy A_ℓ.

Polynomial IRWs/UIRWs. An IRW/UIRW $(\mathbf{T}^\diamond, f, \epsilon)$ is called $(d,k)-$polynomial if for every $\ell \in \mathbf{L}$:

- The set $\mathbf{T}^\diamond_\ell := \mathbf{T}^\diamond \cap (\{\ell\} \times \mathbb{R}^{\mathbf{V}})$ is a closed semi-algebraic set defined by at most k non-strict polynomial inequalities of degree d or less. Equivalently, there exists a set A_ℓ of at most k non-strict polynomial inequalities of degree at most d over $\mathbf{V}$ such that $\nu \models A_\ell$ iff $(\ell, \nu) \in \mathbf{T}^\diamond$.
- The function f_ℓ, defined as $f_\ell(\nu) = f(\ell, \nu)$, is a polynomial of degree at most d over $\mathbf{V}$.

A $(d,k)-$polynomial IRW/UIRW is *explicitly bounded* if each set A_ℓ contains a polynomial inequality $g \geq 0$ such that $\textsc{Sat}(g \geq 0)$ is bounded.

3 Automated Synthesis of IRWs/UIRWs

We now provide our algorithms for synthesizing linear or polynomial witnesses for linear or polynomial systems.

3.1 Synthesizing Linear IRWs/UIRWs

Problem Definition. Given a $k-$linear system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, together with a set τ_ℓ of at most k non-strict linear inequalities at every location $\ell \in \mathbf{L}$, synthesize a $k-$linear IRW/UIRW for the target set $\mathbf{T} := \cup_{\ell \in \mathbf{L}} \{\ell\} \times \textsc{Sat}(\tau_\ell)$ or report that no such IRW/UIRW exists. In the sequel, we assume $\mathbf{V} = \{v_1, \ldots, v_r\}$, and $\mathbf{L} = \{\ell_0, \ldots, \ell_n\}$.

Mathematical Tool. Our approach in this section is based on a well-known theorem in linear programming, called Farkas' Lemma. The presentation we use is similar to [33, 40].

Lemma 1 (Farkas' Lemma [53, 82]). *Consider a set $\mathbf{V} = \{v_1, \ldots, v_r\}$ of real-valued variables and the following system of m linear inequalities over $\mathbf{V}$:*

$$\Phi : \begin{cases} a_{1,0} + a_{1,1} \cdot v_1 + \ldots + a_{1,r} \cdot v_r \geq 0 \\ \vdots \\ a_{m,0} + a_{m,1} \cdot v_1 + \ldots + a_{m,r} \cdot v_r \geq 0 \end{cases}$$

When Φ is satisfiable, it entails a given linear inequality

$$\psi : c_0 + c_1 v_1 + \ldots + c_r v_r \geq 0$$

iff ψ can be written as a non-negative linear combination of $1 \geq 0$ and the inequalities in Φ, i.e. iff there exist non-negative *real numbers $y_0, y_1, \ldots, y_m$, such that: $c_0 = y_0 + \sum_{i=1}^m y_i \cdot a_{i,0}$; $c_1 = \sum_{i=1}^m y_i \cdot a_{i,1}$; $\ldots$; $c_r = \sum_{i=1}^m y_i \cdot a_{i,r}$. Moreover, Φ is unsatisfiable iff $-1 \geq 0$ can be derived as above.*

We often encounter Φ's with both strict and non-strict linear inequalities. So, we use the following corollary:

Corollary 1 (Proof in Appendix D). *Take a set $\mathbf{V} = \{v_1, \ldots, v_r\}$ of real-valued variables and the following system of m linear inequalities over $\mathbf{V}$:*

$$\Phi : \begin{cases} a_{1,0} + a_{1,1} \cdot v_1 + \ldots + a_{1,r} \cdot v_r \bowtie_1 0 \\ \vdots \\ a_{m,0} + a_{m,1} \cdot v_1 + \ldots + a_{m,r} \cdot v_r \bowtie_m 0 \end{cases}$$

in which $\bowtie_i \in \{>, \geq\}$. When Φ is satisfiable, it entails a given non-strict linear inequality $\psi : c_0 + c_1 v_1 + \ldots + c_r v_r \geq 0$ iff ψ can be written as a non-negative linear combination of $1 \geq 0$ and the inequalities in Φ, i.e. iff there exist non-negative *real numbers $y_0, y_1, \ldots, y_m$, such that: $c_0 = y_0 + \sum_{i=1}^m y_i \cdot a_{i,0}$; $c_1 = \sum_{i=1}^m y_i \cdot a_{i,1}$; $\ldots$; $c_r = \sum_{i=1}^m y_i \cdot a_{i,r}$. Moreover, Φ is unsatisfiable iff either $-1 \geq 0$ can be derived as above, or $0 > 0$ can be derived as above while requiring $\sum_{\bowtie_i \in \{>\}} y_i > 0$ (i.e. in order to derive a strict inequality, we should use at least one of the strict inequalities in Φ with non-zero coefficient).*

Overview of the Algorithm. Before presenting our algorithm in detail, we provide a high-level overview of its steps:

- **Step 1.** The algorithm creates a template for the desired IRW/UIRW. It considers every expression that should be synthesized as part of an IRW/UIRW, i.e. the descriptions of $\mathbf{T}^\diamond$ and f, and creates a template for it in which the coefficients are unknown (to be synthesized).
- **Step 2.** The algorithm generates a series of so-called "constraint pairs". These constraint pairs are of a specific form that is amenable to Farkas' Lemma. They encode the requirements that $\mathbf{T}^\diamond$ should be a $\mathbf{T}-$inductive set and that f should be a valid $\mathbf{T}$-ranking function.
- **Step 3.** The algorithm applies Farkas' lemma to the constraints generated in Step 2 and translates them to an equivalent system of quadratic (in)equalities over the unknown *template* variables. After this step, no program variable appears in the quadratic constraints.
- **Step 4.** The algorithm adds a few additional constraints that ensure the existence of a suitable initial valuation.
- **Step 5.** Finally, it solves the constraints by calling an off-the-shelf Quadratic Programming (QP) solver. It then plugs back the solution into the templates generated in Step 1 to obtain the desired IRW/UIRW.

We now provide the details of each step of our algorithm.

Step 1. Setting up a template. Take a $k-$linear IRW/UIRW for reaching $\mathbf{T}$ in S. It has a $k-$linear set $\mathbf{T}^\diamond$, defined by a set A_ℓ of k linear inequalities at every location ℓ, and a function f, also defined by a linear expression f_ℓ at every location ℓ. The algorithm sets up a symbolic *template* for each A_ℓ

$I : x, y \geq 10$
1 : **if** $x < y$:
2 : $\square\ x := x + 10$
3 : $\square\ x := x + 5$
4 :

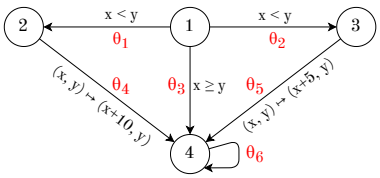

Figure 2. Our Running Example

and f_ℓ. Concretely, it symbolically computes the following expressions, in which the $\widehat{c_{\ell,i,j}}$'s and $\widehat{d_{\ell,j}}$'s are unknowns[†]:

$$\widehat{A_\ell} : \begin{cases} \widehat{c_{\ell,1,0}} + \widehat{c_{\ell,1,1}} \cdot v_1 + \ldots + \widehat{c_{\ell,1,r}} \cdot v_r \geq 0 \\ \vdots \\ \widehat{c_{\ell,k,0}} + \widehat{c_{\ell,k,1}} \cdot v_1 + \ldots + \widehat{c_{\ell,k,r}} \cdot v_r \geq 0 \end{cases}$$

$$\widehat{f_\ell} = \widehat{d_{\ell,0}} + \widehat{d_{\ell,1}} \cdot v_1 + \ldots + \widehat{d_{\ell,r}} \cdot v_r$$

Intuitively, the goal of the algorithm is to find suitable real values for the unknown coefficients (i.e. $\widehat{c_{\ell,i,j}}$'s and $\widehat{d_{\ell,j}}$'s) so that when we plug them into $\widehat{f_\ell}$'s and $\widehat{A_\ell}$'s, they yield a valid IRW/UIRW. The algorithm also defines a new unknown $\widehat{\epsilon}$ which serves as the decrease parameter for f.

Example 6. *Consider the system in Figure 2. We will use this system as our running example and aim to synthesize a 2-linear IRW and a 2-linear UIRW for it. For the IRW case, suppose that the target set is* $\mathbf{T} = \{(4, \nu) \mid \nu \models (x \geq y + 8)\}$. *For the UIRW case, we let* $\mathbf{T}' = \{(4, \nu) \mid \nu \models (x \geq y + 4)\}$. *The algorithm generates a variable* $\widehat{\epsilon}$ *and the following templates:*

$$\widehat{A_1} : \begin{cases} \widehat{c_0} + \widehat{c_1} \cdot x + \widehat{c_2} \cdot y \geq 0 \\ \widehat{c_3} + \widehat{c_4} \cdot x + \widehat{c_5} \cdot y \geq 0 \end{cases} \quad \widehat{A_2} : \begin{cases} \widehat{c_6} + \widehat{c_7} \cdot x + \widehat{c_8} \cdot y \geq 0 \\ \widehat{c_9} + \widehat{c_{10}} \cdot x + \widehat{c_{11}} \cdot y \geq 0 \end{cases}$$

$$\widehat{A_3} : \begin{cases} \widehat{c_{12}} + \widehat{c_{13}} \cdot x + \widehat{c_{14}} \cdot y \geq 0 \\ \widehat{c_{15}} + \widehat{c_{16}} \cdot x + \widehat{c_{17}} \cdot y \geq 0 \end{cases} \quad \widehat{A_4} : \begin{cases} \widehat{c_{18}} + \widehat{c_{19}} \cdot x + \widehat{c_{20}} \cdot y \geq 0 \\ \widehat{c_{21}} + \widehat{c_{22}} \cdot x + \widehat{c_{23}} \cdot y \geq 0 \end{cases}$$

$$\widehat{f_1} = \widehat{d_0} + \widehat{d_1} \cdot x + \widehat{d_2} \cdot y \qquad \widehat{f_2} = \widehat{d_3} + \widehat{d_4} \cdot x + \widehat{d_5} \cdot y$$

$$\widehat{f_3} = \widehat{d_6} + \widehat{d_7} \cdot x + \widehat{d_8} \cdot y \qquad \widehat{f_4} = \widehat{d_9} + \widehat{d_{10}} \cdot x + \widehat{d_{11}} \cdot y$$

The goal is to synthesize values for $\widehat{\epsilon}, \widehat{c_0}, \ldots, \widehat{c_{23}}$ *and* $\widehat{d_0}, \ldots, \widehat{d_{11}}$, *so that the templates above become an IRW/UIRW.*

Step 2a. Computing IRW Constraint Pairs. This step is only performed when we want to synthesize an IRW. In an IRW, the existential inductive set $\mathbf{T}^\diamond$ should satisfy the condition that for every state $\sigma \in \mathbf{T}^\diamond \setminus \mathbf{T}$, there exists a successor $\sigma' \in \mathbf{T}^\diamond$ of σ with $f(\sigma') \leq f(\sigma) - \epsilon$. Let $\ell \in \mathbf{L}$ be a location and Θ_ℓ be the set of transitions out of ℓ, i.e. transitions whose pre-location is ℓ. The IRW properties at ℓ are equivalent to:

$$\forall \nu \in \mathbb{R}^{\mathbf{V}},\ \nu \models \widehat{A_\ell} \Rightarrow \left(\nu \models \tau_\ell \ \vee \ \bigvee_{\theta=(\ell,\ell',\varphi,\mu)\in\Theta_\ell} \xi(\theta)\right) \quad (1)$$

where $\xi(\theta) = \xi(\ell, \ell', \varphi, \mu)$ is defined as:

$$\xi(\theta) := \left(\nu \models \varphi \wedge \mu(\nu) \models \widehat{A_{\ell'}} \wedge \widehat{f_{\ell'}}(\mu(\nu)) \leq \widehat{f_\ell}(\nu) - \widehat{\epsilon}\right) \quad (2)$$

Intuitively, the constraint in (1) says that if $\nu \models A_\ell$ or equivalently $(\ell, \nu) \in \mathbf{T}^\diamond$, then either $(\ell, \nu) \in \mathbf{T}$ which is equivalent to $\nu \models \tau_\ell$, or there exists a transition $\theta \in \Theta_\ell$, using which we can obtain a successor $(\ell', \mu(\nu)) \in \mathbf{T}^\diamond$ such that $f(\ell', \mu(\nu)) \leq f(\ell, \nu) - \epsilon$. The latter is formalized by $\xi(\theta)$. In this step, the algorithm symbolically computes (1) and writes it in the following equivalent format:

$$\forall \nu \in \mathbb{R}^{\mathbf{V}},\ \left(\nu \models \widehat{A_\ell} \wedge \bigwedge_{\theta=(\ell,\ell',\varphi,\mu)} \neg\xi(\theta)\right) \Rightarrow \nu \models \tau_\ell \quad (3)$$

Let P_ℓ be the LHS assertion in (3) above. Then P_ℓ is constructed from logical operations and atomic strict/non-strict linear inequalities over $\mathbf{V}$. The coefficients in these linear inequalities contain the unknown $\widehat{c_{\ell,i,j}}$'s and $\widehat{d_{\ell,j}}$'s.

The algorithm writes P_ℓ in disjunctive normal form, obtaining $P_\ell = P_{\ell,1} \vee P_{\ell,2} \vee \ldots \vee P_{\ell,p}$, where each $P_{\ell,i}$ is a conjunction of linear inequalities over $\mathbf{V}$. It then symbolically computes the following "constraint pair" for every $P_{\ell,i}$:

$$\gamma_{\ell,i} := (P_{\ell,i}, \tau_\ell) \quad (4)$$

The algorithm computes these constraint pairs for every $\ell \in \mathbf{L}$ and stores them in a set Γ. Every constraint pair $\gamma = (\lambda, \varrho) \in \Gamma$ consists of two parts. λ is a set of strict/non-strict linear inequalities, while ϱ is a set of only *non-strict* linear inequalities. Informally, γ encodes the requirement that every inequality in ϱ be entailed by inequalities in λ.

Example 7. *Consider the system in Figure 2 together with the templates generated in Example 6. In this step, the algorithm considers location* $1 \in \mathbf{L}$, *and writes the constraint in (3):*

$$\widehat{A_1} \ \wedge \ \neg\xi(\theta_1) \ \wedge \ \neg\xi(\theta_2) \ \wedge \ \neg\xi(\theta_3) \Rightarrow \tau_1 \quad (5)$$

Intuitively, this constraint says if we are at a $\mathbf{T}^\diamond$ *state in location* 1, *and cannot transition to another* $\mathbf{T}^\diamond$ *with smaller* $\widehat{f}$*-value, in other words* $\neg\xi(\theta_1) \ \wedge \ \neg\xi(\theta_2) \ \wedge \ \neg\xi(\theta_3)$, *then we must already be in a target state (satisfy* τ_1*). There is no target state at location* 1, *so we can assume* $\tau_1 \equiv (-1 \geq 0)$. *The algorithm computes (5) symbolically:*

$$\widehat{c_0} + \widehat{c_1} \cdot x + \widehat{c_2} \cdot y \geq 0 \ \wedge \ \widehat{c_3} + \widehat{c_4} \cdot x + \widehat{c_5} \cdot y \geq 0 \ \wedge$$
$$\neg(x < y \ \wedge \ \widehat{c_6} + \widehat{c_7} \cdot x + \widehat{c_8} \cdot y \geq 0 \ \wedge \widehat{c_9} + \widehat{c_{10}} \cdot x + \widehat{c_{11}} \cdot y \geq 0 \ \wedge$$
$$\widehat{d_3} + \widehat{d_4} \cdot x + \widehat{d_5} \cdot y \leq \widehat{d_0} + \widehat{d_1} \cdot x + \widehat{d_2} \cdot y - \widehat{\epsilon}) \ \wedge$$
$$\neg(x < y \ \wedge \ \widehat{c_{12}} + \widehat{c_{13}} \cdot x + \widehat{c_{14}} \cdot y \geq 0 \ \wedge \ \widehat{c_{15}} + \widehat{c_{16}} \cdot x + \widehat{c_{17}} \cdot y \geq 0 \ \wedge$$
$$\widehat{d_6} + \widehat{d_7} \cdot x + \widehat{d_8} \cdot y \leq \widehat{d_0} + \widehat{d_1} \cdot x + \widehat{d_2} \cdot y - \widehat{\epsilon}) \ \wedge$$
$$\neg(x \geq y \ \wedge \ \widehat{c_{18}} + \widehat{c_{19}} \cdot x + \widehat{c_{20}} \cdot y \geq 0 \ \wedge \ \widehat{c_{21}} + \widehat{c_{22}} \cdot x + \widehat{c_{23}} \cdot y \geq 0 \ \wedge$$
$$\widehat{d_9} + \widehat{d_{10}} \cdot x + \widehat{d_{11}} \cdot y \leq \widehat{d_0} + \widehat{d_1} \cdot x + \widehat{d_2} \cdot y - \widehat{\epsilon})$$
$$\Rightarrow (-1 \geq 0)$$

[†]We use the notation $\widehat{\cdot}$ to denote variables/expressions whose values should be synthesized by the algorithm.

Intuitively, the first part of the constraint above models a state in $\mathbf{T}^\diamond$ at location 1. The second part models the fact that it is not possible to take transition θ_1 and reach another state in $\mathbf{T}^\diamond$ at location 2 such that the $\widehat{f}$-value decreases by at least $\widehat{\epsilon}$. The next two parts model similar constraints for θ_2 and θ_3. Finally, the last part says that if no suitable transition is possible, then the current state must itself be a target, which is impossible in this case because there are no target states at location 1. Next, the algorithm writes the constraint above in DNF:

$$P_{1,1} \ \vee \ P_{1,2} \ \vee \ \ldots \ \vee P_{1,p} \Rightarrow (-1 \geq 0)$$

Just as before, the algorithm computes each of $P_{1,1}, \ldots, P_{1,p}$ concretely in terms of $x, y, \widehat{\epsilon}, \widehat{c_i}$'s and $\widehat{d_i}$'s, but to save space, we omit the full expansion here, e.g. we can assume $P_{1,1}$ is:

$$\widehat{c_0} + \widehat{c_1} \cdot x + \widehat{c_2} \cdot y \geq 0 \ \wedge \ \widehat{c_3} + \widehat{c_4} \cdot x + \widehat{c_5} \cdot y \geq 0 \ \wedge \ x \geq y \wedge$$
$$\widehat{d_9} + \widehat{d_{10}} \cdot x + \widehat{d_{11}} \cdot y > \widehat{d_0} + \widehat{d_1} \cdot x + \widehat{d_2} \cdot y - \widehat{\epsilon}$$

This corresponds to the case where we cannot use either transition θ_1 or θ_2 because $x \geq y$, and also taking transition θ_3 will lead to a state whose $\widehat{f}$-value is not small enough. For each such $P_{1,i}$ it generates a constraint pair $(P_{1,i}, \tau_1) = (P_{1,i}, -1 \geq 0)$. The algorithm handles other locations similarly.

Step 2b. Computing UIRW Constraint Pairs. This step is only performed when synthesizing a UIRW and is similar to its IRW variant in Step 2a above. Due to space constraints, we have relegated the details of this step to Appendix C.

Step 2c. Computing Non-negativity Constraints. In an IRW/UIRW, the ranking function f should have *non-negative* value over $\mathbf{T}^\diamond$. Let $\ell \in \mathbf{L}$. The non-negativity condition at ℓ is equivalent to:

$$\forall \nu \in \mathbb{R}^{\mathbf{V}}, \ \nu \models \widehat{A_\ell} \Rightarrow \widehat{f_\ell}(\nu) \geq 0$$

To ensure this constraint, for every $\ell \in \mathbf{L}$, the algorithm adds the constraint pair $(\widehat{A_\ell}, \widehat{f_\ell} \geq 0)$ to Γ.

Example 8. *Based the templates of Example 6, the algorithm creates the following non-negativity constraint pair $\gamma = (\lambda, \varrho)$, encoding $\lambda \Rightarrow \varrho$, at location $1 \in \mathbf{L}$:*

$$\lambda : \begin{cases} \widehat{c_0} + \widehat{c_1} \cdot x + \widehat{c_2} \cdot y \geq 0 \\ \widehat{c_3} + \widehat{c_4} \cdot x + \widehat{c_5} \cdot y \geq 0 \end{cases} \qquad \varrho : (\widehat{d_0} + \widehat{d_1} \cdot x + \widehat{d_2} \cdot y \geq 0)$$

Step 3. Applying Farkas' Lemma. The algorithm applies Corollary 1 to every constraint pair generated in the previous step to obtain a non-linear constraint system based on the template variables (i.e. $\widehat{c_{\ell,i,j}}$'s and $\widehat{d_{\ell,j}}$'s), the ranking parameter $\widehat{\epsilon}$, and new variables defined in this step. Crucially, this non-linear constraint system does not include any of the variables in $\mathbf{V}$. We now explain this step more concretely.

For every constraint pair $\gamma = (\lambda, \varrho) \in \Gamma$, we know that λ is a set of strict/non-strict linear inequalities $\{\lambda_{i,0} + \vec{\lambda_i} \cdot \vec{\mathbf{V}} \bowtie_i 0\}_{i=1}^m$, in which $\bowtie_i \in \{>, \geq\}$. Moreover, ϱ is a set of *non-strict* inequalities and every inequality in ϱ should be entailed by λ. Let $\alpha_0 + \alpha_1 \cdot v_1 + \ldots + \alpha_r \cdot v_r \geq 0 \equiv \alpha_0 + \vec{\alpha} \cdot \vec{\mathbf{V}} \geq 0$ be an inequality in ϱ. According to Corollary 1, there are three cases in which $\{\lambda_{i,0} + \lambda_i \cdot \mathbf{V} \bowtie_i 0\}_{i=1}^m$ entails $\alpha_0 + \alpha \cdot \mathbf{V} \geq 0$:

(i) $\alpha_0 + \alpha \cdot \mathbf{V} \geq 0$ is a non-negative combination of $1 \geq 0$ and $\{\lambda_{i,0} + \lambda_i \cdot \mathbf{V} \bowtie_i 0\}_{i=1}^m$, or
(ii) $-1 \geq 0$ can be derived as above, or
(iii) $0 > 0$ can be derived as above.

The algorithm writes constraints that model each of the three cases above and then combines them disjunctively. Given their similarity, we only explain (i). The algorithm creates new variables $\widehat{y_0}, \ldots, \widehat{y_m} \geq 0$ and computes the equality

$$\alpha_0 + \alpha \cdot \mathbf{V} = \widehat{y_0} + \textstyle\sum_{i=1}^m \widehat{y_i} \cdot (\lambda_{i,0} + \lambda_i \cdot \mathbf{V}) \tag{6}$$

The two sides above are linear expressions over $\mathbf{V}$. As such, they are equal iff they agree on the coefficient of every term. The algorithm equates the corresponding coefficients, and adds these equalities to the constraint system:

$$\alpha_0 = \widehat{y_0} + \textstyle\sum_{i=1}^m \widehat{y_i} \cdot \lambda_{i,0}$$

i.e. the constant factor should be equal on both sides, and

$$\forall j \neq 0 \qquad \alpha_j = \textstyle\sum_{i=1}^m \widehat{y_i} \cdot \lambda_{i,j}$$

i.e. the coefficient of every variable $v_j \in \mathbf{V}$ should be equal.

The algorithm handles (ii) and (iii) similarly, except that in (iii) we should ensure that at least one strict inequality is used when trying to obtain $0 > 0$. Hence, in this case, the algorithm also adds the extra constraint $\sum_{\bowtie_i \in \{>\}} \widehat{y_i} > 0$.The algorithm performs the same operations for every constraint pair $\gamma = (\lambda, \varrho)$ and every linear inequality in ϱ and combines the resulting non-linear constraint systems conjunctively.

Example 9. *Consider the constraint pair $\gamma = (\lambda, \varrho)$ below, which was obtained in Example 7:*

$$\lambda : \begin{cases} \widehat{c_0} + \widehat{c_1} \cdot x + \widehat{c_2} \cdot y \geq 0 \\ \widehat{c_3} + \widehat{c_4} \cdot x + \widehat{c_5} \cdot y \geq 0 \\ x - y \geq 0 \\ \widehat{d_9} + \widehat{d_{10}} \cdot x + \widehat{d_{11}} \cdot y - \widehat{d_0} - \widehat{d_1} \cdot x - \widehat{d_2} \cdot y + \widehat{\epsilon} > 0 \end{cases} \qquad \varrho : (-1 \geq 0)$$

We want to make sure that λ entails ϱ. By Corollary 1, either ϱ or $-1 \geq 0$ or $0 > 0$ should be a non-negative combination of inequalities in λ. Here, ϱ is itself $-1 \geq 0$, so we take two cases:

- $-1 \geq 0$ is obtainable from λ: *The algorithm creates unknowns $\widehat{y_0}, \widehat{y_1}, \ldots, \widehat{y_4} \geq 0$ and computes this equality:*

$$\widehat{y_0} + \widehat{y_1} \cdot (\widehat{c_0} + \widehat{c_1} \cdot x + \widehat{c_2} \cdot y) + \widehat{y_2} \cdot (\widehat{c_3} + \widehat{c_4} \cdot x + \widehat{c_5} \cdot y) + \widehat{y_3} \cdot (x - y) +$$
$$\widehat{y_4} \cdot (\widehat{d_9} + \widehat{d_{10}} \cdot x + \widehat{d_{11}} \cdot y - \widehat{d_0} - \widehat{d_1} \cdot x - \widehat{d_2} \cdot y + \widehat{\epsilon}) = -1.$$

Our program variables are x and y. All other variables are created by the algorithm and we need to synthesize a value for them. The above is an equality between two polynomials in $\mathbb{R}[x, y]$ that has to hold for all values of x and y. Hence, the algorithm equates its corresponding coefficients:

- $\widehat{y_1} \cdot \widehat{c_1} + \widehat{y_2} \cdot \widehat{c_4} + \widehat{y_3} + \widehat{y_4} \cdot \widehat{d_{10}} - \widehat{y_4} \cdot \widehat{d_1} = 0$ *(for x),*
- $\widehat{y_1} \cdot \widehat{c_2} + \widehat{y_2} \cdot \widehat{c_5} - \widehat{y_3} + \widehat{y_4} \cdot \widehat{d_{11}} - \widehat{y_4} \cdot \widehat{d_2} = 0$ *(for y),*
- $\widehat{y_0} + \widehat{y_1} \cdot \widehat{c_0} + \widehat{y_2} \cdot \widehat{c_3} + \widehat{y_4} \cdot \widehat{d_9} - \widehat{y_4} \cdot \widehat{d_0} + \widehat{y_4} \cdot \widehat{\epsilon} = -1$.

- $0 > 0$ is obtainable from λ: *The algorithm creates* 5 *new variables* $\widehat{y_5}, \ldots, \widehat{y_9}$ *and proceeds to obtain equalities over non-program variables in the exact same manner as in the previous case, except that it also adds the condition* $\widehat{y_9} > 0$.

Step 4. Computing Initial Constraints. An IRW/UIRW should have an initial state (ℓ_0, ν) with $\nu \models I$. Equivalently,

$$\exists \nu_0 = (\nu_{0,1}, \ldots, \nu_{0,r}) \in \mathbb{R}^{\mathbf{V}}, \nu \models \widehat{A_{\ell_0}} \wedge I. \tag{7}$$

By $k-$linearity of the system S, we know that the initial assertion I is a conjunction of at most k linear inequalities. Thus, the assertion above is a conjunction of at most $2k$ linear inequalities, and is equivalent to $\textsc{Sat}(\widehat{A_{\ell_0}} \wedge I) \neq \emptyset$. In this step, the algorithm creates r new variables $\widehat{\nu_{0,1}}, \ldots, \widehat{\nu_{0,r}}$, symbolically computes the linear inequalities in (7), and adds them (conjunctively) to the non-linear constraint system.

Example 10. *For our example (Figure 2), the algorithm creates new variables* $\widehat{\nu_{0,x}}$ *and* $\widehat{\nu_{0,y}}$ *and computes the following:*

$$\widehat{c_0} + \widehat{c_1} \cdot \widehat{\nu_{0,x}} + \widehat{c_2} \cdot \widehat{\nu_{0,y}} \geq 0 \qquad \widehat{\nu_{0,x}} \geq 10$$
$$\widehat{c_3} + \widehat{c_4} \cdot \widehat{\nu_{0,x}} + \widehat{c_5} \cdot \widehat{\nu_{0,y}} \geq 0 \qquad \widehat{\nu_{0,y}} \geq 10$$

The left constraints ensure that the valuation $\widehat{\nu_0} = (\widehat{\nu_{0,x}}, \widehat{\nu_{0,y}})$ *satisfies* $\widehat{A_1}$ *and the right constraints enforce* I.

Step 5. Solving the Resulting Constraint System. Finally, the algorithm uses an off-the-shelf solver to solve the resulting non-linear constraint system. If the system is unsatisfiable, it reports that no $k-$linear IRW/UIRW exists. Otherwise, it obtains a solution $\mathfrak{s}$ of the non-linear constraint system. Let $\mathfrak{s}(\widehat{x})$ denote the value assigned by $\mathfrak{s}$ to variable $\widehat{x}$, and extend this definition in the natural way so to any expression e. The algorithm outputs $A_\ell := \mathfrak{s}(\widehat{A_\ell})$ and $f_\ell := \mathfrak{s}(\widehat{f_\ell})$, for all $\ell \in \mathbf{L}$, as the IRW/UIRW. Moreover, $\mathfrak{s}(\widehat{\nu_{0,1}}, \ldots, \widehat{\nu_{0,r}})$ is the initial state, and $\mathfrak{s}(\widehat{\epsilon})$ is the decrease parameter for f.

Example 11. *When the algorithm solves the non-linear constraints obtainted in the previous steps, it successfully synthesizes the following IRW (left) for* $\mathbf{T} = \{(4, \nu) \mid \nu \models (x \geq y+8)\}$ *and UIRW (right) for* $\mathbf{T}' = \{(4, \nu) \mid \nu \models (x \geq y+4)\}$*:*

ℓ	A_ℓ	f_ℓ
1	$y-2 \leq x \leq y-1$	2
2	$y-2 \leq x \leq y-1$	1
3	$-1 \geq 0$	-1
4	$x \geq y+8$	0

$\epsilon = 1, \quad \nu_0 = (11, 12)$

ℓ	A_ℓ	f_ℓ
1	$y-0.6 \leq x \leq y-0.5$	2
2	$y-0.6 \leq x \leq y-0.5$	1
3	$y-0.6 \leq x \leq y-0.5$	1
4	$x \geq y+4.4$	0

$\epsilon = 1, \quad \nu_0 = (11, 11.55)$

Theorem 3 (Soundness, Proof in Appendix D)**.** *Given a* $k-$*linear system* $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$*, and a* $k-$*linear set* $\mathbf{T}$ *of target states, every solution of the non-linear constraint system solved in Step 5 of the algorithm above produces a valid* $k-$*linear IRW/UIRW for* $\mathbf{T}$ *in* S.

Theorem 4 (Completeness, Proof in Appendix D)**.** *Given a* $k-$*linear system* $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$*, and a* $k-$*linear set* $\mathbf{T}$ *of target states, every* $k-$*linear IRW/UIRW for* $\mathbf{T}$ *in* S *is produced by some solution to the non-linear constraint system solved in Step 5 of the algorithm above.*

Theorem 5 (Complexity, Proof in Appendix D)**.** *For fixed constants* k *and* β*, given a* $k-$*linear* $\beta-$*branching system* $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$*, and a* $k-$*linear set* $\mathbf{T}$ *of target states, Steps 1–4 of the algorithm above lead to a* polynomial-time *reduction from the problem of generating a* $k-$*linear IRW/UIRW to solving a Quadratic Programming (QP) instance.*

3.2 Synthesizing Polynomial IRWs/UIRWs

We now extend our algorithm to the case where the system, the target set, and the IRW/UIRW are all polynomial.

Outline. Our algorithm is similar to the linear case. The main difference is that we should now handle polynomial constraint pairs. In other words, we have entailments of the form $P \rightarrow Q$ in which both P and Q are conjunctions of polynomial inequalities with unknown coefficients. Our goal is to rewrite such entailments as a system of quadratic equations over the unknown coefficients, so that we can use off-the-shelf QP-solvers. To do this, we will need polynomial counterparts of Farkas' Lemma and Corollary 1. Note that Farkas' lemma also provided a direct way of checking unsatisfiability of systems of linear inequalities. We will similarly need an unsatisfiability criterion for the polynomial case, since $P \rightarrow Q$ holds vacuously if P is unsatisfiable. We first introduce and develop these required mathematical tools and then present the algorithm.

Strong Positivity. Let $X \subseteq \mathbb{R}^{\mathbf{V}}$ be a set of valuations and $g \in \mathbb{R}[\mathbf{V}]$ a polynomial over $\mathbf{V}$. We say that g is *strongly* positive over X, and write $X \models g \gg 0$ (or simply $g \gg 0$ when X is clear from context), if $\inf_{x \in X} g(x) > 0$. The real value $\delta := \inf_{x \in X} g(x)$ is called the *positivity gap* of g over X. g is strongly greater than h, denoted $g \gg h$, iff $g - h \gg 0$.

Problem Definition. Given technical constants $\Upsilon_1, \ldots, \Upsilon_4 \in \mathbb{N}$, a $(d, k)-$polynomial system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, together with a set τ_ℓ of at most k *strong* polynomial inequalities of degree at most d at every location $\ell \in \mathbf{L}$, synthesize a $(d, k)-$polynomial IRW/UIRW for a target set $\mathbf{T}$ that satisfies τ_ℓ at every $\ell \in \mathbf{L}$, i.e. $\mathbf{T} \cap (\{\ell\} \times \mathbb{R}^{\mathbf{V}}) \models \tau_\ell$, or report that no such IRW/UIRW exists. The technical constants Υ_i are bounds on the degrees of various polynomials we construct as part of our algorithm. We will soon discuss them more concretely.

Theorem 6 (Putinar's Positivstellensatz [90])**.** *Consider a set* $\mathbf{V} = \{v_1, \ldots, v_r\}$ *of real-valued variables and the following system of* m *polynomial inequalities over* $\mathbf{V}$*:*

$$\Phi : \{g_1(v_1, \ldots, v_r) \geq 0, \quad \ldots, \quad g_m(v_1, \ldots, v_r) \geq 0$$

where $g_1, \ldots, g_m \in \mathbb{R}[\mathbf{V}]$ *are polynomials. If there exists a* g_i *such that the set* $\textsc{Sat}(g_i \geq 0)$ *is compact, and* Φ *entails a given polynomial inequality* $g(v_1, \ldots, v_r) > 0$ *then there exist polynomials* $h_0, h_1, \ldots, h_m \in \mathbb{R}[\mathbf{V}]$ *such that*

$$g = h_0 + \textstyle\sum_{i=1}^{m} h_i \cdot g_i$$

and every h_i *is a sum of squares, i.e.* $h_i = \sum h_{i,j}^2$ *for some polynomials* $h_{i,j} \in \mathbb{R}[\mathbf{V}]$.

This theorem automatically provides a criterion for satisfiability of Φ, given that the inequality $-1 > 0$ is entailed by Φ iff Φ is unsatisfiable. As before, we need a variant of this theorem that can handle strict inequalities in Φ.

Corollary 2. *Take the following system of m polynomial inequalities over $\mathbf{V} = \{v_1, \ldots, v_r\}$:*

$$\Phi : \{g_1(v_1, \ldots, v_r) \bowtie_1 0, \quad \ldots, \quad g_m(v_1, \ldots, v_r) \bowtie_m 0$$

in which every $g_i \in \mathbb{R}[\mathbf{V}]$ is a polynomial and $\bowtie_i \in \{>, \geq\}$. Also, assume that there is some i such that the set $\textsc{Sat}(g_i \geq 0)$ is compact, or equivalently $\textsc{Sat}(g_i \bowtie_i 0)$ is bounded. If Φ is satisfiable, then it entails a strong *polynomial inequality $g(v_1, \ldots, v_r) \gg 0$, iff there exist a constant $y_0 \in (0, \infty)$ and polynomials $h_0, \ldots, h_m \in \mathbb{R}[\mathbf{V}]$ such that*

$$g = y_0 + h_0 + \textstyle\sum_{i=1}^{m} h_i \cdot g_i \tag{8}$$

and every h_i is a sum of squares, i.e. $h_i = \sum h_{i,j}^2$ for some polynomials $h_{i,j} \in \mathbb{R}[\mathbf{V}]$.

Proof. It is obvious that any polynomial g that can be represented as in Equation (8) is strongly positive over $\textsc{Sat}(\Phi)$ and has a positivity gap of at least $y_0 > 0$. Suppose Φ is satisfiable and entails $g \gg 0$ with positivity gap δ, and choose $0 < y_0 < \delta' < \delta$. We have $\textsc{Sat}(\Phi) \subseteq \textsc{Sat}(g > \delta')$ so $\textsc{Sat}(\overline{\Phi}) = \overline{\textsc{Sat}(\Phi)} \subseteq \overline{\textsc{Sat}(g > \delta')} = \textsc{Sat}(g \geq \delta') \subseteq \textsc{Sat}(g > y_0)$. Hence, $\overline{\Phi}$ entails $g - y_0 > 0$. Applying Theorem 6 to $\overline{\Phi}$ and $g - y_0$ leads to the desired result. □

This corollary characterizes strongly positive polynomials over $\textsc{Sat}(\Phi)$. However, it is only applicable when Φ is satisfiable. We are using this assumption in the proof above when we write $\textsc{Sat}(\overline{\Phi}) = \overline{\textsc{Sat}(\Phi)}$. If Φ is unsatisfiable, then it vacuously entails any polynomial inequality, but the corollary above is inapplicable. Therefore, we also need a criterion for unsatisfiability of Φ. We use Hilbert's Strong Nullstellensatz to obtain a suitable criterion (Theorem 8).

Theorem 7 (Strong Nullstellensatz [9])**.** *Take $\mathbf{V} = \{v_1, \ldots, v_r\}$ and let $g_1, \ldots, g_m, g \in \mathbb{R}[\mathbf{V}]$ be polynomials over $\mathbf{V}$. Then exactly one of the following statements holds:*

- *There exists a valuation $\nu \in \mathbb{R}^{\mathbf{V}}$, such that $g_1(\nu) = g_2(\nu) = \ldots = g_m(\nu) = 0$, but $g(\nu) \neq 0$.*
- *There exist $\alpha \in \mathbb{N} \cup \{0\}$ and polynomials $h_1, \ldots, h_m \in \mathbb{R}[\mathbf{V}]$ such that $\sum_{i=1}^{m} h_i \cdot g_i = g^{\alpha}$.*

The following theorem is the main theoretical basis of our synthesis algorithm and characterizes unsatisfiability of Φ when it contains both strict and non-strict inequalities:

Theorem 8. *Take a set $\mathbf{V} = \{v_1, \ldots, v_r\}$ of real-valued variables and a system of m polynomial inequalities over $\mathbf{V}$:*

$$\Phi : \{g_1(v_1, \ldots, v_r) \bowtie_1 0, \quad \ldots, \quad g_m(v_1, \ldots, v_r) \bowtie_m 0$$

where each $g_i \in \mathbb{R}[\mathbf{V}]$ is a polynomial and $\bowtie_i \in \{>, \geq\}$. Φ is unsatisfiable iff at least one of the following statements holds:

(i) There exist a constant $y_0 \in (0, \infty)$ and sum-of-square polynomials $h_0, \ldots, h_m \in \mathbb{R}[\mathbf{V}]$ such that

$$-1 = y_0 + h_0 + \textstyle\sum_{i=1}^{m} h_i \cdot g_i.$$

(ii) There exist $\alpha \in \mathbb{N} \cup \{0\}$ and $h_1, \ldots, h_m \in \mathbb{R}[\mathbf{V}^*]$ for $\mathbf{V}^* = \mathbf{V} \cup \{w_1, \ldots, w_m\}$, such that for some $1 \leq j \leq m$ with $\bowtie_j \in \{>\}$, we have

$$w_j^{2\cdot\alpha} = \textstyle\sum_{i=1}^{m} h_i \cdot (g_i - w_i^2).$$

Proof. If Φ is satisfiable, then it cannot entail $-1 > 0$, so (i) is impossible. We now show that (ii) implies unsatisfiability of Φ as well. Define $\tilde{g}_i(v_1, \ldots, v_r, w_1, \ldots, w_m) := g_i(v_1, \ldots, v_r) - w_i^2$. So, we have $w_j^{2\cdot\alpha} = \sum_{i=1}^{m} h_i \cdot \tilde{g}_i$. Moreover, $g_j^{\alpha} = \left(\tilde{g}_j + w_j^2\right)^{\alpha} = \sum_{i=0}^{\alpha} \binom{\alpha}{i} \tilde{g}_j^i \cdot w_j^{2\cdot(\alpha-i)} = w_j^{2\cdot\alpha} + h'_j \cdot \tilde{g}_j$ for some $h'_j \in \mathbb{R}[\mathbf{V}^*]$. So, letting $h''_i = h_i$ for $i \neq j$ and $h''_j = h_j + h'_j$, we have

$$g_j^{\alpha} = \textstyle\sum_{i=1}^{m} h''_i \cdot (g_i - w_i^2)$$

Let $\nu \in \mathbb{R}^{\mathbf{V}} \cap \textsc{Sat}(\Phi)$. We extend ν to $\nu^* \in \mathbb{R}^{\mathbf{V}^*}$ as follows: for every w_i, let $\nu^*(w_i) = \sqrt{\nu(g_i)}$. So, we have $\nu^*(g_i - w_i^2) = 0$, and hence the RHS of the equation above is 0 at ν^*. On the other hand, we have $\nu^*(g_j^{\alpha}) = \nu(g_j^{\alpha}) = \left(\nu(g_j)\right)^{\alpha} > 0$. This contradiction shows that Φ is unsatisfiable.

We now prove the other side. Suppose that Φ is unsatisfiable. If $\overline{\Phi}$ is unsatisfiable, then it entails $-1.5 > 0$ and hence we can apply Theorem 6 to write $-1.5 = h_0 + \sum_{i=1}^{m} h_i \cdot g_i$ for some sum-of-squares polynomials h_i, which is equivalent to $-1 = 0.5 + h_0 + \sum_{i=1}^{m} h_i \cdot g_i$, hence leading to case (i) above.

The only remaining case is if $\overline{\Phi}$ is satisfiable but Φ is not. Reorder the inequalities in Φ so that the non-strict inequalities appear first. Let j be the smallest index for which $\Phi[1 \ldots j]$, i.e. the set of first j inequalities in Φ, is unsatisfiable. By definition, $\Phi[1 \ldots j-1]$ is satisfiable and hence $\overline{\textsc{Sat}(\Phi[1 \ldots j-1])} = \textsc{Sat}(\overline{\Phi}[1 \ldots j-1])$. Moreover, since $\Phi[1 \ldots j] = \Phi[1 \ldots j-1] \wedge (g_j > 0)$ is unsatisfiable, we know that $\Phi[1 \ldots j-1]$ entails $g_j \leq 0$. In other words, $\textsc{Sat}(\Phi[1 \ldots j-1]) \subseteq \textsc{Sat}(g_j \leq 0)$. Taking closures from both sides shows that $\overline{\Phi}[1 \ldots j-1]$ entails $g_j \leq 0$. So, $\overline{\Phi}[1 \ldots j]$ entails $g_j = 0$. Define $\tilde{g}_i(v_1, \ldots, v_r, w_1, \ldots, w_m) := g_i(v_1, \ldots, v_r) - w_i^2$. We claim there is no valuation $\nu^* \in \mathbb{R}^{\mathbf{V}^*}$ such that for all $1 \leq i \leq j$, $\tilde{g}_i(\nu^*) = 0$, but $g_j(\nu^*) \neq 0$. To prove this, suppose that such a valuation exists, and let ν be its restriction to $\mathbf{V}$. For each $1 \leq i \leq j$, since $\tilde{g}_i(\nu^*) = 0$, we have $g_i(\nu) \geq 0$. Moreover, $g_j(\nu) = g_j(\nu^*) \neq 0$. This is a contradiction with the previously proven fact that $\overline{\Phi}[1 \ldots j]$ entails $g_j = 0$. Applying the Strong Nullstellensatz (Theorem 7) to the $\tilde{g}_i$'s and g_j, we conclude that there exist a non-negative integer α and polynomials $\tilde{h}_1, \ldots, \tilde{h}_j \in \mathbb{R}[\mathbf{V}^*]$ such that $g_j^{\alpha} = \sum_{i=1}^{j} \tilde{h}_i \cdot \tilde{g}_i$. Note that $g_j^{\alpha} = \left(\tilde{g}_j + w_j^2\right)^{\alpha} = \sum_{i=0}^{\alpha} \binom{\alpha}{i} \tilde{g}_j^i \cdot w_j^{2\cdot(\alpha-i)} = w_j^{2\cdot\alpha} + h'_j \cdot \tilde{g}_j$ for some $h'_j \in \mathbb{R}[\mathbf{V}^*]$.

Defining $h_i = \tilde{h}_i$ for all $i \neq j$, and $h_j = \tilde{h}_j - h'_j$, we get

$$w_j^{2\cdot\alpha} = \sum_{i=1}^{j} h_i \cdot \tilde{g}_i = \sum_{i=1}^{j} h_i \cdot (g_i - w_i^2).$$

The Synthesis Algorithm. As in the linear case, our algorithm consists of 5 steps. The main differences are in Steps 1 and 3. In Step 1, our algorithm should now generate a polynomial template. Moreover, in Step 3, it employs Corollary 2 and Theorem 8 for characterizing entailment. The other steps are exactly like our previous algorithm.

Step 1. Setting up a template. The algorithm symbolically computes the set of monomials of degree at most d over $\mathbf{V}$:

$$M_d(\mathbf{V}) := \{\mathfrak{m}_1, \mathfrak{m}_2, \ldots, \mathfrak{m}_u\} :=$$
$$\{v_1^{\alpha_1} \cdot v_2^{\alpha_2} \cdot \ldots \cdot v_r^{\alpha_r} \mid \alpha_1, \ldots, \alpha_r \in \mathbb{N} \cup \{0\} \ \wedge\ \alpha_1 + \ldots + \alpha_r \leq d\}.$$

It sets up templates for A_ℓ and f_ℓ at every $\ell \in \mathbf{L}$:

$$\widehat{A_\ell} : \begin{cases} \widehat{c_{\ell,1,1}} \cdot \mathfrak{m}_1 + \ldots + \widehat{c_{\ell,1,u}} \cdot \mathfrak{m}_u \geq 0 \\ \vdots \\ \widehat{c_{\ell,k,1}} \cdot \mathfrak{m}_1 + \ldots + \widehat{c_{\ell,k,u}} \cdot \mathfrak{m}_u \geq 0 \end{cases}$$

$$\widehat{f_\ell} = \widehat{d_{\ell,1}} \cdot \mathfrak{m}_1 + \ldots + \widehat{d_{\ell,u}} \cdot \mathfrak{m}_u$$

The $\widehat{c_{\ell,i,j}}$'s and $\widehat{d_{\ell,j}}$'s are unknown variables for which we should synthesize a value such that the $\widehat{A_\ell}$'s and $\widehat{f_\ell}$'s form an IRW or a UIRW. We do not need to add a separate constant factor to our templates because $1 \in M_d(\mathbf{V})$.

Step 2. Computing Constraint Pairs. Steps 2a–2c are the same as in Section 3.1. However, note that the resulting constraint pairs $\gamma = (\lambda, \varrho) \in \Gamma$ are now polynomial. Concretely, λ is a set of strict or non-strict polynomial inequalities over $\mathbf{V}$ and ϱ is a set of strong polynomial inequalities over $\mathbf{V}$.

Step 3. Applying the Stellensätze. The algorithm applies Corollary 2 and Theorem 8 to every constraint pair generated in the previous step to obtain a non-linear constraint system based on the template variables, $\widehat{\epsilon}$, and new variables defined in this step. Let $\gamma = (\lambda, \varrho) \in \Gamma$ be a constraint pair. λ is a set of polynomial inequalities of the form $\{g_i \bowtie_i 0\}_{i=1}^m$. Let $g \gg 0$ be a strong polynomial inequality in ϱ. λ must entail $g \gg 0$. The algorithm considers three cases:

(i) *λ is unsatisfiable due to case (i) in Theorem 8*: The algorithm considers the set $M_{\Upsilon_1}(\mathbf{V}) := \{\mathfrak{m}_1, \mathfrak{m}_2, \ldots, \mathfrak{m}_\mathfrak{u}\}$ of all monomials of degree at most Υ_1 over $\mathbf{V}$. Recall that Υ_1 is the first technical parameter given in input. It then generates the following templates $\widehat{h_i}$ for $0 \leq i \leq m$:

$$\widehat{h_i} := \widehat{\eta_{i,1}} \cdot \mathfrak{m}_1 + \ldots + \widehat{\eta_{i,\mathfrak{u}}} \cdot \mathfrak{m}_\mathfrak{u}$$

by introducing new variables $\widehat{\eta_{i,j}}$. It also adds certain constraints on $\widehat{\eta_{i,j}}$'s that ensure every $\widehat{h_i}$ is a sum-of-squares. See Appendix F for more details. Then, the algorithm introduces $\widehat{y_0} > 0$ and symbolically computes

$$-1 = \widehat{y_0} + \widehat{h_0} + \textstyle\sum_{i=1}^m \widehat{h_i} \cdot g_i$$

Finally, the algorithm equates the corresponding coefficients on the two sides of the equality above, and obtains quadratic equalities over the unknown variables. No program variable appears in these equalities.

(ii) *λ is unsatisfiable due to case (ii) in Theorem 8*: The algorithm considers the set $M^*_{\Upsilon_2} := \{\mathfrak{m}^*_1, \ldots, \mathfrak{m}^*_{\mathfrak{u}^*}\}$ of all monomials of degree at most Υ_2 (our second technical parameter) over the extended variable set $\mathbf{V}^* = \mathbf{V} \cup \{w_1, \ldots, w_m\}$. It generates templates $\widehat{h_i}$ for $1 \leq i \leq m$:

$$\widehat{h_i} := \widehat{\eta_{i,1}} \cdot \mathfrak{m}^*_1 + \ldots + \widehat{\eta_{i,\mathfrak{u}}} \cdot \mathfrak{m}^*_{\mathfrak{u}^*}$$

and symbolically computes the equality below for every index j with a *strict* inequality $g_j > 0$ in λ:

$$w_j^{2\cdot\Upsilon_3} = \textstyle\sum_{i=1}^m \widehat{h_i} \cdot (g_i - w_i^2).$$

Here Υ_3 is our third technical parameter and both sides are polynomials in $\mathbb{R}[\mathbf{V}^*]$. The algorithm equates the corresponding coefficients on the LHS and RHS and obtains quadratic equalities over unknown variables. The systems of quadratic equalities generated for each index j are combined *disjunctively*.

(iii) *g is a combination of g_i's as in Corollary 2*: The algorithm considers the set $M_{\Upsilon_4} := \{\mathfrak{m}_1, \ldots, \mathfrak{m}_\mathfrak{u}\}$ of monomials of degree at most Υ_4 over $\mathbf{V}$, and generates the following templates $\widehat{h_i}$ for $0 \leq i \leq m$:

$$\widehat{h_i} := \widehat{h_i} := \widehat{\eta_{i,1}} \cdot \mathfrak{m}_1 + \ldots + \widehat{\eta_{i,\mathfrak{u}}} \cdot \mathfrak{m}_\mathfrak{u}$$

by introducing new variables $\widehat{\eta_{i,j}}$ and adding constraints that ensure every $\widehat{h_i}$ is a sum-of-squares (Appendix F). It then introduces $\widehat{y_0} > 0$ and symbolically computes

$$g = \widehat{y_0} + \widehat{h_0} + \textstyle\sum_{i=1}^m \widehat{h_i} \cdot g_i.$$

Finally, the algorithm translates this equality to quadratic equalities over template variables.

The systems of quadratic equalities generated in (i)–(iii) above are combined disjunctively.

Steps 4 and 5. These steps are the same as in Section 3.1.

Example 12. *Suppose that $\Upsilon_1 = \Upsilon_2 = \Upsilon_3 = \Upsilon_4 = 1$, and the algorithm is in Step 3, handling the following constraint pair:*

$$\lambda : \begin{cases} \widehat{c_1} \cdot x > 0 \\ \widehat{c_2} \cdot y \geq 0 \end{cases} \qquad \varrho : (\widehat{c_3} \cdot x \cdot y + c_4 \gg 0)$$

The algorithm considers the following cases:

(i) It generates three new template polynomials

$$\widehat{h_0} = \widehat{\eta_{0,1}} + \widehat{\eta_{0,2}} \cdot x + \widehat{\eta_{0,3}} \cdot y$$
$$\widehat{h_1} = \widehat{\eta_{1,1}} + \widehat{\eta_{1,2}} \cdot x + \widehat{\eta_{1,3}} \cdot y$$
$$\widehat{h_2} = \widehat{\eta_{2,1}} + \widehat{\eta_{2,2}} \cdot x + \widehat{\eta_{2,3}} \cdot y$$

and computes a quadratic system of (in)equalities over the $\widehat{\eta_{i,j}}$'s that ensures every $\widehat{h_i}$ is a sum-of-squares (See Appendix F for details). The algorithm then computes the following equality symbolically (with $\widehat{y_0} > 0$):

$$-1 = \widehat{y_0} + \widehat{h_0} + \widehat{h_1} \cdot \widehat{c_1} \cdot x + \widehat{h_2} \cdot \widehat{c_2} \cdot y$$

and rewrites it as quadratic equalities between the unknown variables by equating the coefficients of corresponding terms on the two sides of the polynomial equality. Intuitively, if there is a valuation for the unknown variables that satisfies these constraints, then -1 is a combination of $\widehat{c_1} \cdot x, \widehat{c_2} \cdot y$ and sum-of-square polynomials. Hence, λ is unsatisfiable.

(ii) The algorithm creates two new program variables w_1, w_2 and sets up the following templates:

$$\widehat{h_3} = \widehat{\eta_{3,1}} + \widehat{\eta_{3,2}} \cdot x + \widehat{\eta_{3,3}} \cdot y + \widehat{\eta_{3,4}} \cdot w_1 + \widehat{\eta_{3,5}} \cdot w_2$$
$$\widehat{h_4} = \widehat{\eta_{4,1}} + \widehat{\eta_{4,2}} \cdot x + \widehat{\eta_{4,3}} \cdot y + \widehat{\eta_{4,4}} \cdot w_1 + \widehat{\eta_{4,5}} \cdot w_2$$

Unlike the previous case, $\widehat{h_3}$ and $\widehat{h_4}$ need not be sum-of-squares. It then writes the equality:

$$w_1^2 = \widehat{h_3} \cdot (\widehat{c_1} \cdot x - w_1^2) + \widehat{h_4} \cdot (\widehat{c_2} \cdot y - w_2^2),$$

and converts this polynomial equality to quadratic equalities by equating corresponding coefficients. Note that the LHS and RHS of the polynomial equality above are in $\mathbb{R}[x, y, w_1, w_2]$. By Theorem 8, any solution to these constraints serves as a proof for unsatisfiability of λ.

(iii) The algorithm generates the following templates:

$$\widehat{h_5} = \widehat{\eta_{5,1}} + \widehat{\eta_{5,2}} \cdot x + \widehat{\eta_{5,3}} \cdot y,$$
$$\widehat{h_6} = \widehat{\eta_{6,1}} + \widehat{\eta_{6,2}} \cdot x + \widehat{\eta_{6,3}} \cdot y,$$
$$\widehat{h_7} = \widehat{\eta_{7,1}} + \widehat{\eta_{7,2}} \cdot x + \widehat{\eta_{7,3}} \cdot y;$$

enforces them to be sum-of-squares just as in case (i) above (Appendix F) and writes the polynomial equality:

$$\widehat{c_3} \cdot x \cdot y + \widehat{c_4} = \widehat{y_1} + \widehat{h_5} + \widehat{h_6} \cdot \widehat{c_1} \cdot x + \widehat{h_7} \cdot \widehat{c_2} \cdot y$$

in which $\widehat{y_1} > 0$. It handles it similarly to the previous cases. This is again a polynomial equality in $\mathbb{R}[x, y]$.

The algorithm combines the systems of quadratic inequality in (i)–(iii) above disjunctively.

It is now easy to obtain the following theorems, whose proofs are similar to the linear case:

Theorem 9 (Soundness). *Given a $(d, k)-$polynomial system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, and a set τ_ℓ of at most k polynomial inequalities of degree d or less at every $\ell \in \mathbf{L}$, every solution of the non-linear constraint system solved in Step 5 of the algorithm above produces a valid (d, k)-polynomial IRW/UIRW for a target set $\mathbf{T}$ that satisfies τ_ℓ at every $\ell \in \mathbf{L}$.*

Theorem 10 (Semi-completeness). *Take a $(d, k)-$polynomial system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$ and a set τ_ℓ of at most k strong polynomial inequalities of degree d or less at every $\ell \in \mathbf{L}$. Let $W = (\mathbf{T}^\diamond, f, \epsilon)$ be an explicitly bounded $(d, k)-$polynomial IRW/UIRW for a target set $\mathbf{T}$ that satisfies τ_ℓ at every $\ell \in \mathbf{L}$. If large enough values are assigned to technical constants $\Upsilon_1, \ldots, \Upsilon_4$, the witness W is produced by some solution of the non-linear constraint system solved in Step 5 of the algorithm.*

Theorem 11 (Complexity). *For fixed constants k, d and β, and technical constants $\Upsilon_1, \ldots, \Upsilon_4$, given a $(k, d)-$polynomial $\beta-$branching system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, and a set τ_ℓ of at most k polynomial inequalities of degree d or less at every $\ell \in \mathbf{L}$, Steps 1–4 of the algorithm above lead to a* polynomial-time *reduction from the problem of generating a $(k, d)-$polynomial IRW/UIRW to solving a QP instance.*

Remark 2. *Theorem 10 provides* semi-*completeness, i.e. completeness when the chosen technical constants are large enough, because the stellensätze have no bound on the degree of polynomials in their characterizations, but we have to fix a degree in our algorithm. We use $\Upsilon_1, \ldots, \Upsilon_4$ for this purpose. Such results arise routinely in constraint-based termination analysis [34] and invariant generation [56]. In practice, solutions are often found with small technical constants (see Section 4).*

Remark 3. *We presented our approach for* conjunctive *IRWs, i.e. at each label, the set of states that are in $\mathbf{T}^\diamond$ was defined by a* conjunction *of linear/polynomial inequalities. However, the approach can be extended to handle disjunction, as well. In the linear case, this can be achieved using Motzkin's Transposition Theorem [19, 85]. In the polynomial case, we can apply Stengle's Positivstellensatz [99] instead of Putinar's. In both cases, the completeness will be unaffected but the runtime will increase.*

4 Experimental Results

Implementation. We implemented our algorithms for IRW synthesis in Python using SymPy [84] for symbolic computations. We also added several heuristics for improving performance. Notably, we used Z3 [49] to identify and discard unsatisfiable or tautological constraint pairs. The QPs were then solved by LOQO [102]. All results were obtained on an Intel Core i5-2540M (2.6 GHz) machine with 8 GB RAM running Ubuntu, with a limit of 1800 seconds per task.

Linear Benchmarks. We used C benchmarks from SV-COMP 2020 [22]. We considered tasks in the "Reachability/Safety" category of the competition in which the error states were reachable. Of these, we removed benchmarks that cannot be soundly modeled as transition systems with real variables, i.e. benchmarks containing pointers, arrays, etc. This left us with 30 benchmarks. The only source of non-determinism in these benchmarks is their input. We handle this by synthesizing an initial valuation as part of the IRW. In absence of integer division/mod, our approach can soundly handle integer variables by requiring that the synthesized initial valuation (in Step 4) is integral, i.e. $\widehat{v_{0,i}}$'s are integer unknowns. This leads to a mixed QP instance, but there are very few integral variables, and the problem remains tractable.

Polynomial Benchmarks. For the polynomial case, all standard benchmarks focused on safety. Hence, we created 6 simple programs with complex reachability structure to showcase the strengths of our approach. Due to space restrictions, these examples are put in Appendix G. Specifically, it is noteworthy that these benchmarks demonstrate the fact that our algorithm's success is not dependent on the length or proportion of paths that reach the target set $\mathbf{T}$.

Benchmark	|L|	|Θ|	|V|	k	|QP|	Gen	Solve	Ours	CPAchecker	VeriAbs	Dangerzone	Depth
gcnr2008	8	14	4	2	1838	14.8	81.4	96.2	**1.8**	17.6	3.3	4
trex02-2	5	7	1	2	260	1.8	3.4	5.3	4.3	16.6	**2.4**	3
nec11	4	8	3	2	2871	13.2	45.7	58.9	4.2	10.8	**2.7**	6
terminator_02-1	5	8	3	3	1962	12.0	19.4	31.4	4.2	17.2	**2.9**	5
simple_2-2	3	5	1	2	267	1.7	1.5	**3.2**	4.3	10.4	T/O	3
count_up_down-2	3	4	3	2	244	1.6	1.9	**3.5**	4.4	5.9	T/O	4
nested_1b	4	7	1	2	819	3.3	11.0	14.3	4.7	**3.5**	T/O	9
while_infinite_loop_4	10	14	1	2	1223	5.1	7.9	13.0	**4.1**	15.3	N/A	6
multivar_1-2	3	6	2	2	900	5.8	19.8	25.6	**4.4**	9.0	T/O	4
trex01-1	14	27	6	3	9491	69.7	228.2	297.8	4.5	17.3	**4.5**	7
for_bounded_loop1	10	13	5	2	1579	9.8	30.1	39.9	5.6	16.8	**4.6**	12
underapprox_1-1	3	6	2	2	901	3.7	5.6	9.3	**4.7**	14.0	T/O	21
sum03-1	9	14	2	2	20963	77.9	413.1	491.0	6.1	16.3	**4.8**	57
underapprox_2-1	3	6	2	2	901	3.7	14.5	18.2	**4.9**	10.6	T/O	21
sum04-1	6	10	2	2	1082	5.4	8.0	13.4	**5.1**	17.0	T/O	22
terminator_03-1	6	11	2	3	1740	10.0	25.7	35.6	**5.1**	9.7	T/O	5
trex03-1	4	12	6	2	8500	49.2	197.9	247.0	**5.2**	9.1	5.3	5
Mono1_1-1	3	5	1	2	262	1.3	4.4	**5.7**	T/O	377.2	T/O	5.50e+7
sum01_bug02_base	7	13	3	2	7972	38.0	133.4	171.4	**6.0**	17.3	24.1	18
id_trans	5	11	5	2	11192	68.7	171.2	239.8	**6.4**	19.8	10.8	11
sum01_bug02	7	12	4	3	17632	60.0	218.6	278.6	**6.5**	17.3	71.8	26
sum01-1	7	12	3	2	7316	36.9	55.1	92.0	**7.6**	16.7	T/O	32
nested_1-2	4	6	2	2	329	2.9	8.0	**10.9**	T/O	86.0	N/A	6.17e+9
Mono5_1	5	7	3	4	1048	8.1	31.8	39.8	T/O	332.4	**19.6**	3.50e+7
const_1-2	3	6	2	2	901	4.6	17.6	**22.2**	T/O	49.6	T/O	3076
Mono3_1	6	8	2	3	660	4.0	20.0	**24.0**	T/O	369.9	T/O	4.00e+6
Mono4_1	5	7	2	4	949	5.3	22.1	**27.3**	T/O	635.8	T/O	3.50e+6
Mono6_1	5	7	3	5	1502	11.6	48.5	**60.1**	T/O	382.2	T/O	3.50e+7
deep_nested	7	17	5	5	3686	28.6	69.6	**98.2**	T/O	T/O	N/A	1.46e+48
nested_delay_notd2	6	9	6	3	12858	55.3	396.7	451.9	T/O	**247.4**	N/A	1385

Table 1. Experimental Results over SV-COMP Benchmarks. Times are in seconds. "Gen"/"Solve" are the times spent generating/solving the QP. "Ours" is our total runtime. "T/O" denotes timeout. "Depth" is the length of the shortest path that reaches the target states (bugs). Since our approach does not output a path, but rather an IRW, we calculated the depths manually.

Benchmark	|L|	|Θ|	|V|	k	d	|QP|	Gen	Solve	Ours	CPAchecker	VeriAbs	Dangerzone	Depth
sqrt2	5	7	2	5	2	2494	24.1	22.4	46.4	10.5	19.7	**2.7**	207
sqrt1	3	4	2	4	2	920	10.7	30.1	40.8	T/O	207.3	**2.7**	633
robot2	5	8	4	5	2	5537	71.1	681.7	752.9	T/O	F	**3.1**	29982
robot1	5	8	4	5	2	5537	69.8	724.2	794.0	T/O	F	**5.97**	2127
sum	3	4	3	5	2	1826	20.4	59.7	**80.1**	T/O	F	T/O	30003
sum2	3	4	3	5	3	2476	36.8	167.5	**204.2**	T/O	T/O	T/O	30003

Table 2. Experimental Results over Polynomial Programs. 'F' denotes failure. We set our ϒ variables equal to d.

Previous Tools. We compare our approach against the two best-performing tools in the Reachability/Safety category of SV-COMP 2020, namely VeriAbs [2] and CPAchecker [25], as well as the most related previous work, Dangerzone [48].

Linear Results. The results over linear benchmarks are summarized in Table 1. Our approach could handle every linear reachability benchmark in SV-COMP 2020. It is noteworthy that according to the SV-COMP results, none of the participating model checkers could handle all benchmarks of Table 1. CPAchecker times out on 9 of the instances, whereas VeriAbs fails on only 1 instance. Additionally, Dangerzone is not applicable to 4 of the instances and times out on 14. By manual inspection, we realized that CPAChecker and VeriAbs are faster than our approach when reachability can be attained using liberal abstractions and a relatively short path (benchmarks towards the top of Table 1). This is not surprising, given that in these situations, abstract interpretation and symbolic execution are considerably faster than quadratic constraint solving. However, as the paths become longer and sparser (towards the the bottom of Table 1), the advantages of our approach begin to show. When the paths are long, i.e. thousands of steps of program execution, CPAchecker always fails to verify the instance. VeriAbs manages to handle these instances by a combination of ideas from loop pruning, loop summarization, abstract interpretation and bounded model checking. However, this comes with a considerable overhead, leading to a much worse performance in comparison with our approach. Finally, Dangerzone, while being able to efficiently find some of the deep bugs, fails on other instances. This is also not surprising, given that Dangerzone does not provide completeness guarantees and relies on evolutionary methods in its synthesis process.

Our runtime is mainly dependent on the size of the QP instance, which in turn depends on the size of the program

and the degree of the IRW. In contrast, previous approaches' runtimes depend on the depth of the bug. Some previous approaches, e.g. VeriAbs, also perform abstraction, so their runtime becomes dependent on the number of refinements needed to detect the bug. Informally speaking, other approaches are faster in detecting shallow and wide bugs, whereas ours performs better on deep and narrow bugs. Of course, our performance also depends on the QP solver.

Nested Loop Benchmark. The program below illustrates the core of `deep-nested`, the only linear benchmark that could be handled by neither VeriAbs nor CPAchecker:

```
for (a := 0; a < M − 1; a := a + 1):
  for (b := 0; b < M − 1; b := b + 1):
    for (c := 0; c < M − 1; c := c + 1):
      for (d := 0; d < M − 1; d := d + 1):
        for (e := 0; e < M − 1; e := e + 1):
          if M − 2 ≤ a, b, c, d, e:
            print("target reached")
```

In this program, M is the largest value that fits into a 32-bit integer. We also ran these tools over this benchmark with an extended time limit of 12 hours, but they timed out. Moreover, according to SV-COMP results, no other participating model checker could handle this example, either. We believe this is because the target state can only be reached after an enormously-long path. Moreover, the target set is quite thin and even the smallest loss of precision in abstraction can cause a failure to prove reachability. However, this particular benchmark is not at all challenging for our method. The runtime of our method does not depend on the length of the paths, and we do not perform abstraction. Moreover, our approach is complete for linear IRWs. As such, it can easily prove reachability in this program.

Polynomial Results. Table 2 shows our experimental results over 6 polynomial instances. Informally, `sqrt1` is a simple program that given an input integer $n \geq 1$ computes $s = \lfloor \sqrt{n} \rfloor$ by trying every possible integer starting from 1. The goal is to (choose a value for n so as to) reach a state with $n - s > 10^5$. `sqrt2` is a more clever variant of the same program that doubles the current value in a single step when the doubled value does not exceed $\lfloor \sqrt{n} \rfloor$. `sum` is a program that sums up all the integers from 1 to n. The goal is to synthesize a value for n such that the sum falls in a specific interval. `sum2` is a similar benchmark in which the program sums squares of all integers from 1 to n. In `robot1`, two robots are put in the same position in a 2d plane. At each step, each robot non-deterministically chooses to move one unit either upwards or to the right. The goal is to reach a state where the square of the distance between the robots is more than 10^5. In `robot2`, the same two robots are placed on the lower-right and upper-left corners of a square of side length 10^4. The goal is to show that they can reach a distance of less than 10 from each other. See Appendix G for details.

As in the linear case, we observe that CPAchecker and VeriAbs can handle cases where the path reaching the targets is short, and when there is no combinatorial explosion in the number of paths due to repeated nondeterministic choice. Notably, CPAchecker can handle `sqrt2` but not `sqrt1`. The only difference between these two programs is that `sqrt2` is more efficient and hence the path to targets is shorter. We also observe that the various other techniques used by VeriAbs, which made it more successful in the linear case, do not extend well to the polynomial case. In contrast, our approach and Dangerzone are able to handle reachability with long paths. Our approach is the only one that can handle all of the polynomial benchmarks. This is due to its semi-completeness over polynomial IRWs.

5 Conclusion and Future Work

We proposed IRWs for reachability analysis over imperative programs. Our approach synthesizes an under-approximation of the set of program states that can reach the target, then uses a ranking argument to ensure eventual reachability. We proved that our approach is sound and complete when there is no restriction over the form of inductive reachability witnesses, and presented automated sound and semi-complete algorithms for synthesizing linear and polynomial inductive reachability witnesses. In practice, our experimental results show that our automated approaches can solve instances beyond the reach of previous methods.

An interesting future direction is to incorporate more advanced ranking-function synthesis methods such as lexicographic ranking functions [5, 18, 26] into reachability analysis. Another direction is to consider how our approach can be extended to automate the search for proofs in incorrectness logic [86]. Invariant generation has been successfully used in automating aspects of Hoare logic and termination analysis [34, 35, 41, 88]. Given that our witnesses are natural duals of inductive invariants, we expect that this direction will be fruitful.

Acknowledgments

This research was partially supported by the ERC CoG 863818 (ForM-SMArt), the National Natural Science Foundation of China (NSFC) Grant No. 61802254, the Huawei Innovation Research Program, the Facebook PhD Fellowship Program, and DOC Fellowship No. 24956 of the Austrian Academy of Sciences (ÖAW).

References

[1] Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Bui Phi Diep. 2016. Counter-Example Guided Program Verification. In *FM*.

[2] Mohammad Afzal, Supratik Chakraborty, Avriti Chauhan, Bharti Chimdyalwar, Priyanka Darke, Ashutosh Gupta, Shrawan Kumar, Charles Babu M, Divyesh Unadkat, and R. Venkatesh. 2020. VeriAbs : Verification by Abstraction and Test Generation (Competition Contribution). In *TACAS*.

[3] Aws Albarghouthi, Arie Gurfinkel, and Marsha Chechik. 2012. From Under-Approximations to Over-Approximations and Back. In *TACAS*.

[4] Aws Albarghouthi, Yi Li, Arie Gurfinkel, and Marsha Chechik. 2012. UFO: A Framework for Abstraction- and Interpolation-Based Software Verification. In *CAV*.

[5] Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord. 2010. Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs. In *SAS*.

[6] Rajeev Alur and David L. Dill. 1990. Automata For Modeling Real-Time Systems. In *ICALP*.

[7] Rajeev Alur, Alon Itai, Robert P. Kurshan, and Mihalis Yannakakis. 1995. Timing Verification by Successive Approximation. *Information and Computation* 118, 1 (1995).

[8] Mohamed Faouzi Atig and Pierre Ganty. 2011. Approximating Petri Net Reachability Along Context-free Traces. In *FSTTCS*.

[9] Michael Francis Atiyah and Ian Grant Macdonald. 1969. *Introduction to Commutative Algebra*. Taylor and Francis.

[10] Roberto Bagnara, Patricia M. Hill, Elisa Ricci, and Enea Zaffanella. 2003. Precise Widening Operators for Convex Polyhedra. In *SAS*.

[11] Roberto Bagnara, Enric Rodríguez-Carbonell, and Enea Zaffanella. 2005. Generation of Basic Semi-algebraic Invariants Using Convex Polyhedra. In *SAS*.

[12] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT Press.

[13] Alexey Bakhirkin and Nir Piterman. 2016. Finding recurrent sets with backward analysis and trace partitioning. In *TACAS*.

[14] Felice Balarin and Alberto L. Sangiovanni-Vincentelli. 1993. An Iterative Approach to Language Containment. In *CAV*.

[15] Thomas Ball, Vladimir Levin, and Sriram K. Rajamani. 2011. A decade of software model checking with SLAM. *Commun. ACM* 54, 7 (2011).

[16] Thomas Ball and Sriram K. Rajamani. 2002. The SLAM project: debugging system software via static analysis. In *POPL*.

[17] Amir M. Ben-Amram and Samir Genaim. 2015. Complexity of Bradley-Manna-Sipma Lexicographic Ranking Functions. In *CAV*.

[18] Amir M. Ben-Amram and Samir Genaim. 2017. On Multiphase-Linear Ranking Functions. In *CAV*.

[19] Adi Ben-Israel. 2017. Motzkin transposition theorem. *Encyclopedia of Mathematics* (2017). http://encyclopediaofmath.org/index.php?title=Motzkin_transposition_theorem

[20] Josh Berdine, Nikolaj Bjørner, Samin Ishtiaq, Jael E. Kriener, and Christoph M. Wintersteiger. 2013. Resourceful Reachability as HORN-LA. In *LPAR*.

[21] Tewodros A Beyene, Corneliu Popeea, and Andrey Rybalchenko. 2013. Solving existentially quantified horn clauses. In *CAV*.

[22] Dirk Beyer. 2020. Advances in automatic software verification: SV-COMP 2020. In *TACAS*.

[23] Dirk Beyer, Adam Chlipala, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. 2004. Generating Tests from Counterexamples. In *ICSE*.

[24] Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. 2007. The software model checker Blast. *International Journal on Software Tools for Technology Transfer* 9, 5-6 (2007).

[25] Dirk Beyer and M. Erkan Keremoglu. 2011. CPAchecker: A Tool for Configurable Software Verification. In *CAV*.

[26] Aaron R Bradley, Zohar Manna, and Henny B Sipma. 2005. Linear ranking with reachability. In *CAV*.

[27] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. 2005. Termination of Polynomial Programs. In *VMCAI*.

[28] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. 1990. Symbolic Model Checking: 10^{20} States and Beyond. In *LICS*.

[29] Cristian Cadar and Koushik Sen. 2013. Symbolic execution for software testing: three decades later. *Commun. ACM* 56, 2 (2013).

[30] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. 2014. The nuXmv Symbolic Model Checker. In *CAV*.

[31] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *CAV*.

[32] Aleksandar Chakarov and Sriram Sankaranarayanan. 2014. Expectation Invariants for Probabilistic Program Loops as Fixed Points. In *SAS*.

[33] Krishnendu Chatterjee, Hongfei Fu, Amir Goharshady, and Nastaran Okati. 2018. Computational approaches for stochastic shortest path on succinct MDPs. In *IJCAI*, Vol. 2018.

[34] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2016. Termination analysis of probabilistic programs through Positivstellensatz's. In *CAV*.

[35] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2019. Non-polynomial worst-case analysis of recursive programs. *TOPLAS* 41, 4 (2019).

[36] Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Ehsan Kafshdar Goharshady. 2020. Polynomial Invariant Generation for Non-deterministic Recursive Programs. In *PLDI*.

[37] Hong-Yi Chen, Byron Cook, Carsten Fuhs, Kaustubh Nimkar, and Peter O'Hearn. 2014. Proving nontermination via safety. In *TACAS*.

[38] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. 2000. Counterexample-Guided Abstraction Refinement. In *CAV*.

[39] Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick Bloem. 2018. *Handbook of model checking*. Springer.

[40] Michael A Colón, Sriram Sankaranarayanan, and Henny B Sipma. 2003. Linear invariant generation using non-linear constraint solving. In *CAV*.

[41] Michael A Colón and Henny B Sipma. 2001. Synthesis of linear ranking functions. In *TACAS*.

[42] Patrick Cousot. 2005. Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming. In *VMCAI*.

[43] Patrick Cousot. 2019. On fixpoint/iteration/variant induction principles for proving total correctness of programs with denotational semantics. In *LOPSTR*.

[44] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*.

[45] Christoph Csallner, Nikolai Tillmann, and Yannis Smaragdakis. 2008. DySy: dynamic symbolic execution for invariant inference. In *ICSE*.

[46] Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. 2019. The reachability problem for Petri nets is not elementary. In *STOC*.

[47] Philippe Darondeau, Stéphane Demri, Roland Meyer, and Christophe Morvan. 2012. Petri Net Reachability Graphs: Decidability Status of First Order Properties. *Logical Methods in Computer Science* 8, 4 (2012).

[48] Cristina David, Pascal Kesseli, Daniel Kroening, and Matt Lewis. 2016. Danger invariants. In *FM*.

[49] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *TACAS*.

[50] Edsko de Vries and Vasileios Koutavas. 2011. Reverse Hoare Logic. In *SEFM*.

[51] Isil Dillig, Thomas Dillig, Boyang Li, and Kenneth L. McMillan. 2013. Inductive invariant generation via abductive inference. In *OOPSLA*.

[52] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. 2019. Scaling static analyses at Facebook. *Commun. ACM* 62, 8 (2019).

[53] Julius Farkas. 1902. Theorie der einfachen Ungleichungen. *Journal für die reine und angewandte Mathematik* 124 (1902).

[54] Azadeh Farzan and Zachary Kincaid. 2015. Compositional Recurrence Analysis. In *FMCAD*.

[55] Paul Feautrier and Laure Gonnord. 2010. Accelerated Invariant Generation for C Programs with Aspic and C2fsm. *Electronic Notes in Theoretical Computer Science* 267, 2 (2010).

[56] Yijun Feng, Lijun Zhang, David N Jansen, Naijun Zhan, and Bican Xia. 2017. Finding polynomial loop invariants for probabilistic programs. In *ATVA*.

[57] Robert W Floyd. 1993. Assigning meanings to programs. In *Program Verification*.

[58] Pierre Ganty, Radu Iosif, and Filip Konečnỳ. 2013. Underapproximation of procedure summaries for integer programs. In *TACAS*.

[59] Thomas Martin Gawlitza and David Monniaux. 2012. Invariant Generation through Strategy Iteration in Succinctly Represented Control Flow Graphs. *Logical Methods in Computer Science* 8, 3 (2012).

[60] Roberto Giacobazzi and Francesco Ranzato. 1997. Completeness in Abstract Interpretation: A Domain Perspective. In *AMAST*.

[61] Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. 2000. Making abstract interpretations complete. *Journal of the ACM* 47, 2 (2000).

[62] Patrice Godefroid. 2007. Compositional dynamic test generation. In *POPL*.

[63] Amir Kafshdar Goharshady. 2021. *Parameterized and algebro-geometric advances in static program analysis*. Ph.D. Dissertation.

[64] Laure Gonnord and Peter Schrammel. 2014. Abstract acceleration in linear relation analysis. *Science of Computer Programming* 93 (2014).

[65] Arie Gurfinkel, Ou Wei, and Marsha Chechik. 2006. YASM: A Software Model-Checker for Verification and Refutation. In *CAV*.

[66] Ákos Hajdu and Zoltán Micskei. 2019. Efficient strategies for CEGAR-based model checking. *Journal of Automated Reasoning* (2019).

[67] David Handelman. 1988. Representing polynomials by positive linear functions on compact convex polyhedra. *Pacific J. Math.* 132, 1 (1988).

[68] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. 2002. Lazy abstraction. In *POPL*.

[69] Nicholas J Higham. 2009. Cholesky factorization. *Wiley Interdisciplinary Reviews: Computational Statistics* 1, 2 (2009).

[70] Charles Antony Richard Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (1969).

[71] Gerard J. Holzmann. 1997. The Model Checker SPIN. *IEEE Transactions on Software Engineering* 23, 5 (1997).

[72] Roger Horn and Charles Johnson. 1990. *Matrix Analysis*. Cambridge University Press.

[73] Mingzhang Huang, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. 2019. Modular verification for almost-sure termination of probabilistic programs. In *OOPSLA*.

[74] Joxan Jaffar, Vijayaraghavan Murali, Jorge A. Navas, and Andrew E. Santosa. 2012. TRACER: A Symbolic Execution Tool for Verification. In *CAV*.

[75] Zachary Kincaid, Jason Breck, Ashkan Forouhi Boroujeni, and Thomas W. Reps. 2017. Compositional recurrence analysis revisited. In *PLDI*.

[76] Zachary Kincaid, John Cyphert, Jason Breck, and Thomas W. Reps. 2018. Non-linear reasoning for invariant synthesis. In *POPL*.

[77] Daniel Kroening, Matt Lewis, and Georg Weissenbacher. 2015. Under-approximating loops in C programs for fast counterexample detection. *FMSD* 47, 1 (2015).

[78] Jan Leike and Matthias Heizmann. 2014. Ranking Templates for Linear Loops. In *TACAS*.

[79] Kumar Madhukar, Peter Schrammel, and Mandayam Srivas. 2017. Compositional Safety Refutation Techniques. In *ATVA*.

[80] Rupak Majumdar and Koushik Sen. 2007. Hybrid Concolic Testing. In *ICSE*.

[81] Zohar Manna and Amir Pnueli. 2012. *Temporal verification of reactive systems: safety*. Springer.

[82] Jiri Matousek and Bernd Gärtner. 2007. *Understanding and using linear programming*. Springer.

[83] Ernst W. Mayr. 1981. An Algorithm for the General Petri Net Reachability Problem. In *STOC*.

[84] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (2017).

[85] TS Motzkin. 1936. Beitrage zur Theorie der linearen Ungleichungen (Dissertation, Basel 1933).

[86] Peter W. O'Hearn. 2020. Incorrectness logic. In *POPL*.

[87] Amir Pnueli. 1977. The Temporal Logic of Programs. In *FOCS*.

[88] Andreas Podelski and Andrey Rybalchenko. 2004. A Complete Method for the Synthesis of Linear Ranking Functions. In *VMCAI*.

[89] Andreas Podelski and Andrey Rybalchenko. 2004. Transition Invariants. In *LICS*.

[90] Mihai Putinar. 1993. Positive polynomials on compact semi-algebraic sets. *Indiana University Mathematics Journal* 42, 3 (1993).

[91] Francesco Ranzato. 2013. Complete Abstractions Everywhere. In *VMCAI*.

[92] Henry Gordon Rice. 1953. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.* 74, 2 (1953).

[93] Xavier Rival. 2005. Understanding the Origin of Alarms in Astrée. In *SAS*.

[94] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. 2004. Non-linear loop invariant generation using Gröbner bases. In *POPL*.

[95] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. 2004. Constraint-Based Linear-Relations Analysis. In *SAS*.

[96] David A. Schmidt. 2007. A calculus of logical relations for over- and underapproximating static analyses. *Science of Computer Programming* 64, 1 (2007).

[97] Rahul Sharma and Alex Aiken. 2014. From Invariant Checking to Invariant Inference Using Randomized Search. In *CAV*.

[98] Gagandeep Singh, Markus Püschel, and Martin T. Vechev. 2017. Fast polyhedra abstract domain. In *POPL*.

[99] Gilbert Stengle. 1974. A nullstellensatz and a positivstellensatz in semialgebraic geometry. *Math. Ann.* 207, 2 (1974).

[100] Toru Takisaka, Yuichiro Oyabu, Natsuki Urabe, and Ichiro Hasuo. 2018. Ranking and Repulsing Supermartingales for Reachability in Probabilistic Programs. In *ATVA*.

[101] Caterina Urban, Samuel Ueltschi, and Peter Müller. 2018. Abstract interpretation of CTL properties. In *SAS*.

[102] Robert J Vanderbei. 1999. LOQO: An interior point code for quadratic programming. *Optimization methods and software* 11, 1-4 (1999).

A Universal Reachability Witnesses

$I : i = s = 0 \ \wedge \ n \geq 0$
$a :$ **while** $i \leq n :$
$b :$ $\quad (s, i) := (s + 1, i + 1)$
$c :$ $\quad \square\ (s, i) := (s + 2, i + 1)$
$d :$

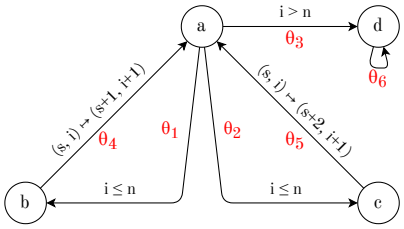

Figure 3. A Non-deterministic Program and its Representation as a Transition System

In this section, we provide detailed definitions and an example of the Universal Inductive Reachability Witnesses (UIRWs).

Universal* T-*inductive Sets. Given a set $\mathbf{T} \subseteq \Sigma$ of target states, a set $\mathbf{T}^\diamond \subseteq \Sigma$ is called *universally* T-*inductive* if for every $\sigma \in \mathbf{T}^\diamond \setminus \mathbf{T}$ and *every* successor σ' of σ, we also have $\sigma' \in \mathbf{T}^\diamond$.

The idea behind universal T-inductive sets is that any execution of the program that starts in such a set $\mathbf{T}^\diamond$ will either reach $\mathbf{T}$ or one can prove using induction that it will never leave $\mathbf{T}^\diamond$, no matter how the non-determinism is resolved.

Universal* T-*ranking Functions. Given a universal T-inductive set $\mathbf{T}^\diamond$, a function $f : \mathbf{T}^\diamond \to [0, \infty)$ is called a *universal* T-*ranking function* with parameter $\epsilon > 0$, if for every $\sigma \in \mathbf{T}^\diamond \setminus \mathbf{T}$ and *every successor* σ' of σ, we have $f(\sigma') \leq f(\sigma) - \epsilon$.

Universal Inductive Reachability Witnesses (UIRWs). Given a set $\mathbf{T}$ of target states in a system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, a *Universal Inductive Reachability Witness* for $\mathbf{T}$ is a tuple $(\mathbf{T}^\diamond, f, \epsilon)$ such that:

- $\mathbf{T}^\diamond$ is a universal T-inductive set;
- $\epsilon \in (0, \infty)$;
- $f : \mathbf{T}^\diamond \to [0, \infty)$ is a universal T-ranking function with parameter ϵ;
- There exists a valuation $\nu \in \mathbb{R}^{\mathbf{V}}$ such that $(\ell_0, \nu) \in \mathbf{T}^\diamond$ and $\nu \models I$.

Example 13. *Figure 3 shows a simple program together with its representation as a transition system. Let* $\mathbf{T} = \{(d, \nu) \mid \nu(s) \geq 20\}$, *i.e. the target is reaching point d with an s value of more than* 20. *Let* $\mathbf{T}^\diamond := \{(\ell, \nu) \mid \nu \models A_\ell\}$ *and* $f(\ell, \nu) := f_\ell(\nu)$ *be defined as follows:*

ℓ	A_ℓ	f_ℓ
a	$n \geq 50 \ \wedge \ s \geq i \geq 0 \ \wedge \ n + 1 \geq i$	$n + 1.5 - i$
b	$n \geq 50 \ \wedge \ s, n \geq i \geq 0$	$n + 1 - i$
c	$n \geq 50 \ \wedge \ s, n \geq i \geq 0$	$n + 1 - i$
d	$s \geq 50$	0

It is easy to check that $(\mathbf{T}^\diamond, f, 0.5)$ *is a UIRW for* $\mathbf{T}$. *Intuitively, this guarantees that if a run starts with an initial valuation that satisfies* A_a, *it will definitely reach a target state.*

B Proofs of Theorems Presented in Section 2

In this section we provide proofs of our basic soundness and completeness theorems.

Theorem 1 (Soundness). Let $\mathbf{T} \subseteq \Sigma$ be a set of states in the system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$.

(i) If there exists an IRW $(\mathbf{T}^\diamond, f, \epsilon)$ for $\mathbf{T}$, then $\mathbf{T}$ is existentially reachable.
(ii) If there exists a UIRW $(\mathbf{T}^\diamond, f, \epsilon)$ for $\mathbf{T}$, then $\mathbf{T}$ is universally reachable.

Proof. We handle each case separately.

(i) We construct a run of S that visits $\mathbf{T}$. By definition of IRW, there exists a state $\sigma_0 = (\ell_0, \nu_0) \in \mathbf{T}^\diamond$ such that $\nu_0 \models I$. We start our run with σ_0 and inductively find the next transitions and states as follows: when we are in a state $\sigma_i \in \mathbf{T}^\diamond$, either (a) $\sigma_i \in \mathbf{T}$ in which case the path until this point has already reached $\mathbf{T}$ and we can extend it to an arbitrary run, or (b) $\sigma_i \in \mathbf{T}^\diamond \setminus \mathbf{T}$, in which case there exists a successor $\sigma_{i+1} \in \mathbf{T}^\diamond$ of σ_i such that $f(\sigma_{i+1}) \leq f(\sigma_i) - \epsilon$, and we transition to σ_{i+1}. Using this procedure, it is not possible to avoid case (a) forever, because each application of (b) decreases the value of f by at least ϵ and f is bounded from below. Hence, the constructed run will reach $\mathbf{T}$.
(ii) We choose $\sigma_0 = (\ell_0, \nu_0)$ as in the previous case. We now prove that every path of length $n := 1 + \lceil f(\sigma_0)/\epsilon \rceil$ starting from σ_0 will reach $\mathbf{T}$. Let $\mathrm{r} = \sigma_0, \theta_0, \sigma_1, \theta_1, \ldots, \sigma_n$ be such a path. If no σ_i is in $\mathbf{T}$, then by definition of universal T-inductiveness, every σ_i is in $\mathbf{T}^\diamond \setminus \mathbf{T}$. So, for each i, we have $f(\sigma_{i+1}) \leq f(\sigma_i) - \epsilon$. Therefore, $f(\sigma_n) \leq f(\sigma_0) - n \cdot \epsilon = f(\sigma_0) - \epsilon - \lceil f(\sigma_0)/\epsilon \rceil \cdot \epsilon < 0$ which is a contradiction because f can only take non-negative values.

□

Theorem 2 (Completeness). Let $\mathbf{T} \subseteq \Sigma$ be a set of states in the system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$.

(i) If $\mathbf{T}$ is existentially reachable, then there exists an IRW $(\mathbf{T}^\diamond, f, \epsilon)$ for $\mathbf{T}$.
(ii) If $\mathbf{T}$ is universally reachable, then there exists a UIRW $(\mathbf{T}^\diamond, f, \epsilon)$ for $\mathbf{T}$.

Proof. In each case, we construct the required IRW/UIRW.

(i) Given that $\mathbf{T}$ is reachable, by definition there exists a path $\pi = (\ell_0, \nu_0), \theta_0, \ldots, (\ell_n, \nu_n)$ such that $(\ell_n, \nu_n) \in \mathbf{T}$ and $\nu_0 \models I$. Without loss of generality, we choose such a π that is prefix-minimal, i.e. that no prefix of π has the same properties. Let $\mathbf{T}^\diamond = \{(\ell_i, \nu_i) | 0 \leq i \leq n\}$, then $\mathbf{T}^\diamond$ is $\mathbf{T}$-inductive, because $(\ell_n, \nu_n) \in \mathbf{T}$ and for every $i \neq n$, the state (ℓ_i, ν_i) can be succeeded by (ℓ_{i+1}, ν_{i+1}). Let $f : \mathbf{T}^\diamond \to [0, \infty)$ be defined as follows: $f(\ell_i, \nu_i) := n - i$. It is easy to verify that $(\mathbf{T}^\diamond, f, 1)$ is an IRW for $\mathbf{T}$.

(ii) We define $\Sigma_k \subseteq \Sigma$ as the set of all states such that every semi-path of length k starting in these states is guaranteed to visit $\mathbf{T}$. Note that $\Sigma_0 = \mathbf{T}$ and if $\sigma \in \Sigma_k \setminus \mathbf{T}$, then by definition every successor σ' of σ must be in Σ_{k-1}. Let $\mathbf{T}^\diamond = \bigcup_{i=0}^{\infty} \Sigma_k$, and for every $\sigma \in \mathbf{T}^\diamond$, define $f(\sigma) := \min\{k \mid \sigma \in \Sigma_k\}$. It is easy to prove by definition-chasing that $(\mathbf{T}^\diamond, f, 1)$ is a UIRW.

□

C Computing Universal Constraint Pairs

In this section, we provide a detailed description of Step 2b of our algorithm, which aims to generate constraint pairs for universal inductive reachability witnesses.

Step 2b. Computing UIRW Constraint Pairs. This step is only performed when synthesizing a UIRW and is similar to its IRW variant in Step 2a. In a UIRW, the universal $\mathbf{T}$-inductive set $\mathbf{T}^\diamond$ should satisfy the condition that for every state $\sigma \in \mathbf{T}^\diamond \setminus \mathbf{T}$, *every* successor σ' of σ is also in $\mathbf{T}^\diamond$. Moreover, given that f is a universal $\mathbf{T}$−ranking function, we must have $f(\sigma') \leq f(\sigma) - \epsilon$ for every such σ'.

Let $\ell \in \mathbf{L}$ be a location. The UIRW properties at ℓ are equivalent to:

$$\forall \nu \in \mathbb{R}^{\mathbf{V}},\ \nu \models \widehat{A_\ell} \Rightarrow \left(\nu \models \tau_\ell \vee \bigwedge_{\theta=(\ell,\ell',\varphi,\mu)} \zeta(\theta)\right) \quad (9)$$

where $\zeta(\theta) = \zeta(\ell, \ell', \varphi, \mu)$ is defined as:

$$\zeta(\theta) := \left(\nu \models \varphi \Rightarrow \left(\mu(\nu) \models \widehat{A_{\ell'}} \wedge \widehat{f_{\ell'}}(\mu(\nu)) \leq \widehat{f_\ell}(\nu) - \widehat{\epsilon}\right)\right) \quad (10)$$

Informally, the constraint in (9) says that if $\nu \models A_\ell$ or equivalently $(\ell, \nu) \in \mathbf{T}^\diamond$, then either $(\ell, \nu) \in \mathbf{T}$, i.e. $\nu \models \tau_\ell$, or *for every transition* θ from ℓ the assertion $\xi(\theta)$ holds, i.e. if the transition is possible ($\nu \models \varphi$), then the successor state $(\ell', \mu(\nu))$ is also in $\mathbf{T}^\diamond$, and the f value decreases by at least ϵ when going to this successor. As in the previous case, the algorithm computes (9) symbolically and writes it in the following equivalent format:

$$\forall \nu \in \mathbb{R}^{\mathbf{V}}, \left(\nu \models \widehat{A_\ell} \wedge \bigvee_{\theta=(\ell,\ell',\varphi,\mu)} \neg\zeta(\theta)\right) \Rightarrow \nu \models \tau_\ell \quad (11)$$

Let Q_ℓ be the LHS assertion above. Similar to Step 2a, Q_ℓ is constructed form logical operations and atomic strict/non-strict linear inequalities over $\mathbf{V}$, and its coefficients include the unknown template variables $\widehat{c_{\ell,i,j}}$'s and $\widehat{d_{\ell,j}}$'s defined in Step 1. The algorithm writes Q_ℓ in disjunctive normal form, hence obtaining $Q_\ell = Q_{\ell,1} \vee Q_{\ell,2} \vee \ldots \vee Q_{\ell,q}$ in which each $Q_{\ell,i}$ is a conjunction of strict/non-strict linear inequalities over $\mathbf{V}$. It then computes the following constraint pair symbolically:

$$\gamma'_{\ell,i} := (Q_{\ell,i}, \tau_\ell)$$

The algorithm performs these operations for every location $\ell \in \mathbf{L}$ and stores all the resulting $\gamma'_{\ell,i}$ constraint pairs in a set Γ.

Example 14. *In our running example (Figure 2), we are looking for a linear UIRW for the target set* $\mathbf{T}' = \{(4, \nu) \mid \nu \models (x \geq y + 4)\}$. *In this step, the algorithm creates constraints at every location. We now demonstrate how the process works for location* 3. *At location* 3, *the algorithm considers*

$$\widehat{A_3} \wedge \neg\zeta(\theta_5) \Rightarrow \tau_3$$

and symbolically computes it as:

$$\widehat{c_{12}} + \widehat{c_{13}} \cdot x + \widehat{c_{14}} \cdot y \geq 0 \wedge \widehat{c_{15}} + \widehat{c_{16}} \cdot x + \widehat{c_{17}} \cdot y \geq 0 \wedge$$
$$\neg(1 \geq 0 \Rightarrow (\widehat{c_{18}}+5\cdot\widehat{c_{19}}+\widehat{c_{19}}\cdot x+\widehat{c_{20}}\cdot y \geq 0 \wedge \widehat{c_{21}}+5\cdot\widehat{c_{22}}+\widehat{c_{22}}\cdot x+\widehat{c_{23}}\cdot y \geq 0 \wedge$$
$$\widehat{d_9} + 5 \cdot \widehat{d_{10}} + \widehat{d_{10}} \cdot x + \widehat{d_{11}} \cdot y \leq \widehat{d_6} + \widehat{d_7} \cdot x + \widehat{d_8} \cdot y - \widehat{\epsilon}))$$
$$\Rightarrow (-1 \geq 0)$$

Note that the transition θ_5 *is unconditional, as such we can assume that its condition is simply* $1 \geq 0$. *Similarly, because there is no target state at location* 3, *we assume* $\tau_3 \equiv (-1 \geq 0)$. *Moreover, the transition* θ_5 *updates the value of* x *to* $x + 5$. *This is taken into account when generating the constraint above. The algorithm writes the LHS of the constraint in DNF and handles it exactly as in Example 7.*

D Proofs of Theorems Presented in Section 3.1

Notation. Given a set $X \subseteq \mathbb{R}^{\mathbf{V}}$, we write $\overline{X}$ to denote the closure of X, i.e. the smallest closed subset of $\mathbb{R}^{\mathbf{V}}$ that contains X. Similarly, for Φ defined as below, we use the notation $\overline{\Phi}$ to denote the system of linear inequalities obtained by replacing every $\bowtie_i$ in Φ with $\geq$.

Corollary 1. Consider a set $\mathbf{V} = \{v_1, \ldots, v_r\}$ of real-valued variables and the following system of m linear inequalities over $\mathbf{V}$:

$$\Phi : \begin{cases} a_{1,0} + a_{1,1} \cdot v_1 + \ldots + a_{1,r} \cdot v_r \bowtie_1 0 \\ \vdots \\ a_{m,0} + a_{m,1} \cdot v_1 + \ldots + a_{m,r} \cdot v_r \bowtie_m 0 \end{cases}$$

in which $\bowtie_i \in \{>, \geq\}$. When Φ is satisfiable, it entails a given non-strict linear inequality

$$\psi : c_0 + c_1 v_1 + \ldots + c_r v_r \geq 0$$

iff ψ can be written as a non-negative linear combination of $1 \geq 0$ and the inequalities in Φ, i.e. iff there exist *non-negative* real numbers $y_0, y_1, \ldots, y_m$, such that:

$$c_0 = y_0 + \sum_{i=1}^{m} y_i \cdot a_{i,0}, c_1 = \sum_{i=1}^{m} y_i \cdot a_{i,1}, \ldots, c_r = \sum_{i=1}^{m} y_i \cdot a_{i,r}.$$

Moreover, Φ is unsatisfiable iff either $-1 \geq 0$ can be derived as above, or $0 > 0$ can be derived as above with the extra requirement that $\sum_{\bowtie_i \in \{>\}} y_i > 0$ (i.e. in order to derive a strict inequality, we should use at least one of the strict inequalities in Φ with non-zero coefficient).

Proof. For the first part, suppose that ψ is entailed by Φ, hence $\{x \in \mathbb{R}^{\mathbf{V}} \mid c_0 + c_1 \cdot x_1 + \ldots + c_r \cdot x_r \geq 0\} \supseteq \{x \in \mathbb{R}^{\mathbf{V}} \mid x \models \Phi\}$. The former is a closed set, hence it also includes the closure of the latter, which is the set of points that satisfy $\overline{\Phi}$. Hence, we can apply Lemma 1 to $\overline{\Phi}$ and ψ to obtain the desired result.

For the second part, if Φ is satisfiable, then obviously no non-negative combination of inequalities in Φ can sum up to a contradiction such as $0 > 0$ or $-1 \geq 0$. If $\overline{\Phi}$ is not satisfiable, then by Lemma 1, we can obtain $-1 \geq 0$. The only remaining case is if $\overline{\Phi}$ is satisfiable but Φ is not. Reorder the inequalities in Φ so that the non-strict inequalities appear first. Then, consider the smallest i for which the first i inequalities in Φ are unsatisfiable. Let $\Phi[1 \ldots i]$ denote the first i inequalities. Based on our ordering, we know that the i−th inequality is strict and of the form $a_{i,0} + a_{i,1} \cdot v_1 + \ldots + a_{i,r} \cdot v_r > 0$. Given that $\Phi[1 \ldots i]$ is unsatisfiable, we know that $\{x \in \mathbb{R}^{\mathbf{V}} \mid x \models \Phi[1 \ldots i-1]\} \subseteq \{x \in \mathbb{R}^{\mathbf{V}} \mid a_{i,0} + a_{i,1} \cdot v_1 + \ldots + a_{i,r} \cdot v_r \leq 0\}$. Therefore, by the first part above, we can write

$$a_{i,0} + a_{i,1} \cdot v_1 + \ldots + a_{i,r} \cdot v_r \leq 0$$

as a non-negative combination of the first $i-1$ inequalities. Moreover, the i−th inequality is:

$$a_{i,0} + a_{i,1} \cdot v_1 + \ldots + a_{i,r} \cdot v_r > 0$$

Summing up these two, we get $0 > 0$. □

Theorem 3 (Soundness). Given an input k−linear system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, and a k−linear set $\mathbf{T}$ of target states, every solution of the non-linear constraint system solved in Step 5 of the algorithm in Section 3.1 produces a valid k−linear IRW/UIRW for $\mathbf{T}$ in S.

Proof. Every solution $\mathfrak{s}$ satisfies the constraints generated in Step 3. Therefore, for every constraint pair $\gamma = (\lambda, \varrho) \in \Gamma$ generated in Step 2 and inequality $\alpha_0 + \alpha \cdot \mathbf{V} \geq 0$ in ϱ, either $\mathfrak{s}(\lambda)$ is unsatisfiable, i.e. a non-negative linear combination of its inequalities sums up to $0 \geq 1$ or $0 > 0$, or there is such a linear combination that sums up to $\alpha_0 + \alpha \cdot \mathbf{V} \geq 0$. In each case, the coefficients of the combination are given by $\mathfrak{s}(\widehat{y_i})$ for the corresponding $\widehat{y_i}$ variables. Moreover, no matter which case happens, the inequalities in ϱ are entailed by λ. By definition, the constraint pairs generated in Step 2 modeled inductivity, non-negativity and ranking conditions and hence $\mathfrak{s}$ satisfies these properties. Finally, $\mathfrak{s}$ satisfies the constraints generated in Step 4. Therefore, we have $\mathfrak{s}(\widehat{\nu_{0,1}}, \ldots, \widehat{\nu_{0,r}}) \models \mathfrak{s}(\widehat{A_{\ell_0}}) \wedge I$. So, all the requirements for IRW/UIRW are met. □

Theorem 4 (Completeness). Given a k−linear system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, and a k−linear set $\mathbf{T}$ of target states, every k−linear IRW/UIRW for $\mathbf{T}$ in S is produced by some solution to the non-linear constraint system solved in Step 5 of the algorithm in Section 3.1.

Proof. We construct the required solution. Let $(\mathbf{T}^\diamond, f, \epsilon)$ be a k−linear IRW/UIRW for $\mathbf{T}$ in S. Let A_ℓ be the set of inequalities defining $\mathbf{T}^\diamond \cap (\ell \times \mathbb{R}^{\mathbf{V}})$, and f_ℓ the linear expression defining f at ℓ. We use the coefficients in A_ℓ's and f_ℓ's as the corresponding values for $\mathfrak{s}(\widehat{c_{\ell,i,j}})$'s and $\mathfrak{s}(\widehat{d_{\ell,j}})$'s. Moreover, we let $\mathfrak{s}(\widehat{\epsilon}) = \epsilon$.

By definition, $\mathbf{T}^\diamond$ is an existential/universal $\mathbf{T}$−inductive set, and f is an existential/universal $\mathbf{T}$−ranking function with parameter ϵ. Therefore, A_ℓ's and f_ℓ's satisfy the constraint pairs generated at Step 2 of the algorithm. By Corollary 1, there are suitable values for each variable $\widehat{y_i}$ such that the constraints in Step 3 are satisfied. We use these values as $\mathfrak{s}(\widehat{y_i})$. Finally, by definition of IRW/UIRW, there exists a valuation $\overline{\nu} \in \mathbb{R}^{\mathbf{V}}$ such that $\overline{\nu} \models A_{\ell_0} \wedge I = \mathfrak{s}(\widehat{A_{\ell_0}}) \wedge I$. We let $\mathfrak{s}(\widehat{\nu_{0,i}}) = \overline{\nu}_i$. It is easy to verify that $\mathfrak{s}$ is a solution to the system of non-linear constraints solved in Step 5. □

Theorem 5 (Complexity). For fixed constants k and β, given a k−linear β−branching system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, and a k−linear set $\mathbf{T}$ of target states, Steps 1–4 of the algorithm in Section 3.1 lead to a *polynomial-time* reduction from the problem of generating a k−linear IRW/UIRW to solving a Quadratic Programming (QP) instance.

Proof. It is easy to verify that all steps of the algorithm run in polynomial time[‡], and that all the generated (in)equalities over non-program variables are quadratic. However, these (in)equalities are not always combined conjunctively. Specifically, in Step 3, the constraints corresponding to cases (i)–(iii) are combined disjunctively. This being said, we can perform the following actions to obtain a QP instance in polynomial time:

- We first convert every inequality of the form $\mathfrak{e} \bowtie 0$ to $\mathfrak{e} - \widehat{x_\mathfrak{e}} = 0$ by introducing a new variable $\widehat{x_\mathfrak{e}} \bowtie 0$.
- We rewrite every disjunction $\mathfrak{e}_1 = 0 \vee \mathfrak{e}_2 = 0$ as $\mathfrak{e}_1 \cdot \mathfrak{e}_2 = 0$. Note that this might create polynomial equalities of higher degree.
- We eliminate terms of degree more than 2 by defining new variables that are equal to their proper divisors, e.g. we rewrite $\widehat{c_1} \cdot \widehat{c_2} \cdot \widehat{c_3}^2$ as $\widehat{v_1} \cdot \widehat{v_2}$ where $\widehat{v_1}, \widehat{v_2}$ are new variables, and add the equalities $\widehat{v_1} = \widehat{c_1} \cdot \widehat{c_2}$ and $\widehat{v_2} = \widehat{c_3}^2$.

The steps above lead to a polynomial blow-up in the size of the system, given that in Step 3 of the algorithm we have disjunctions of at most 3 different boolean formulas. □

[‡]The reason for fixing k and β is to avoid exponential blow-up when rewriting boolean expressions in DNF.

E Synthesizing Linear IRWs/UIRWs for Linear Systems with Polynomial Target Sets

For technical reasons, we need the concept of strong positivity.

Strong Positivity. Let $X \subseteq \mathbb{R}^{\mathbf{V}}$ be a set of valuations and $g \in \mathbb{R}[\mathbf{V}]$ a polynomial over $\mathbf{V}$. We say that g is *strongly* positive over X, and write $X \models g \gg 0$ (or simply $g \gg 0$ when X is clear from context), if $\inf_{x \in X} g(x) > 0$. The real value $\delta := \inf_{x \in X} g(x)$ is called the *positivity gap* of g over X. Moreover, g is strongly greater than h, denoted $g \gg h$, iff $g - h \gg 0$.

Problem Definition. In this section, we consider the following problem: Given a $k-$linear system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$ together with a set τ_ℓ of at most k *strong* polynomial inequalities of degree at most d at every location $\ell \in \mathbf{L}$, synthesize a $k-$linear IRW/UIRW for a target set $\mathbf{T}$ that satisfies τ_ℓ at every $\ell \in \mathbf{L}$, or report that no such IRW/UIRW exists.

Mathematical Tool. Our main mathematical tool in this section is a theorem, due to Handelman, that characterizes positive polynomials over compact polyhedra. Before presenting this theorem, it is useful to define the notion of monoid.

Monoid. Consider a set $\mathbf{V} = \{v_1, \ldots, v_r\}$ of real-valued variables and the following system of m linear inequalities over $\mathbf{V}$:

$$\Phi : \begin{cases} a_{1,0} + a_{1,1} \cdot v_1 + \ldots + a_{1,r} \cdot v_r \bowtie_1 0 \\ \vdots \\ a_{m,0} + a_{m,1} \cdot v_1 + \ldots + a_{m,r} \cdot v_r \bowtie_m 0 \end{cases}$$

in which $\bowtie_i \in \{>, \geq\}$. Let g_i be the LHS of the i-th inequality, i.e. $g_i(v_1, \ldots, v_r) := a_{i,0} + a_{i,1} \cdot v_1 + \ldots + a_{i,r} \cdot v_r$. The monoid of Φ is defined as:

$$\textsc{Monoid}(\Phi) := \left\{ \textstyle\prod_{i=1}^{m} g_i^{\kappa_i} \;\middle|\; \forall 1 \leq i \leq m,\ \kappa_i \in \mathbb{N} \cup \{0\} \right\}.$$

Basically, $\textsc{Monoid}(\Phi)$ is the set of all polynomials that can be obtained by multiplying the linear expressions on the LHS of Φ together. Note that each such expression can appear zero or multiple times in the multiplication. Specifically, it is noteworthy that $1 \in \textsc{Monoid}(\Phi)$.

Theorem 12 ([67]). *Consider a set $\mathbf{V} = \{v_1, \ldots, v_r\}$ of real-valued variables and the following system of m linear inequalities over $\mathbf{V}$:*

$$\Phi : \begin{cases} a_{1,0} + a_{1,1} \cdot v_1 + \ldots + a_{1,r} \cdot v_r \geq 0 \\ \vdots \\ a_{m,0} + a_{m,1} \cdot v_1 + \ldots + a_{m,r} \cdot v_r \geq 0 \end{cases}$$

If Φ is satisfiable, $\textsc{Sat}(\Phi)$ is compact, and Φ entails a given polynomial inequality $g(v_1, \ldots, v_r) > 0$ then there exist reals $y_1, y_2, \ldots, y_u \in [0, \infty)$ and $h_1, h_2, \ldots, h_u \in \textsc{Monoid}(\Phi)$ such that:

$$g = \textstyle\sum_{i=1}^{u} y_i \cdot h_i.$$

As was the case with Farkas' Lemma, it is useful to have a variant of Handelman's theorem that can handle strict inequalities in Φ. We present such a variant, which is a direct corollary of Theorem 12 and characterizes strongly positive polynomials over bounded, but not necessarily closed, polyhedra:

Corollary 3. *Consider a set $\mathbf{V} = \{v_1, \ldots, v_r\}$ of real-valued variables and the following system of m linear inequalities over $\mathbf{V}$:*

$$\Phi : \begin{cases} a_{1,0} + a_{1,1} \cdot v_1 + \ldots + a_{1,r} \cdot v_r \bowtie_1 0 \\ \vdots \\ a_{m,0} + a_{m,1} \cdot v_1 + \ldots + a_{m,r} \cdot v_r \bowtie_m 0 \end{cases}$$

in which $\bowtie_i \in \{>, \geq\}$. If Φ is satisfiable and $\textsc{Sat}(\Phi)$ is bounded, *then Φ entails a given strong polynomial inequality $g(v_1, \ldots, v_r) \gg 0$, or in other words $\textsc{Sat}(\Phi) \models g(v_1, \ldots, v_r) \gg 0$, if and only if there exist constants $y_0 \in (0, \infty)$ and $y_1, y_2, \ldots, y_u \in [0, \infty)$, and polynomials $h_1, h_2, \ldots, h_u \in \textsc{Monoid}(\Phi)$ such that:*

$$g = y_0 + \sum_{i=1}^{u} y_i \cdot h_i. \tag{12}$$

Proof. It is obvious that every g in the form of (12) is strongly positive over $\textsc{Sat}(\Phi)$, given that Φ trivially entails $g \geq y_0 > 0$. We now prove the other side. Suppose Φ entails $g \gg 0$. Let $\delta > 0$ be the positivity gap of g over $\textsc{Sat}(\Phi)$ and choose δ', y_0 such that $0 < y_0 < \delta' < \delta$. So, $\textsc{Sat}(\Phi) \subseteq \textsc{Sat}(g > \delta')$ and hence $\textsc{Sat}(\overline{\Phi}) = \overline{\textsc{Sat}(\Phi)} \subseteq \overline{\textsc{Sat}(g > \delta')} = \textsc{Sat}(g \geq \delta')$. Therefore, $\overline{\Phi}$ entails $g - \delta' \geq 0$. So, it also entails $g - y_0 > 0$. Applying Theorem 12 to $\overline{\Phi}$ and $g - y_0$, we have:

$$g - y_0 = \sum_{i=1}^{u} y_i \cdot h_i$$

which is equivalent to Equation (12). □

The Synthesis Algorithm. Our synthesis algorithm is similar to the one in Section 3.1 and consists of five steps. The main difference is in Step 3, in which constraint pairs are translated to non-linear constraints over template variables. In the previous section, our main tool for this translation was Farkas' Lemma. In this section, due to the more complicated nature of our target sets, we now supplement Farkas' Lemma with Handelman's theorem. For brevity, we do not repeat the presentation of other steps, which are the same as our previous algorithm.

Recall that Step 2 (either Steps 2a and 2c for IRWs, or Steps 2b and 2c for UIRWs) has already generated a set Γ of constraint pairs. Each constraint pair $\gamma \in \Gamma$ is of the form $\gamma = (\lambda, \varrho)$ and encodes the requirement that every inequality in ϱ should be entailed by λ. Moreover, λ is a set of strict/non-strict linear inequalities over $\mathbf{V}$, whereas ϱ is a set of *strong* polynomial inequalities of degree at most d. Let $g \gg 0$ be a strong inequality in ϱ. Either λ is satisfiable and g should be represented in the form of Equation 12 (cf. Corollary 3)

or λ is unsatisfiable, in which case $-1 \geq 0$ or $0 > 0$ can be derived as non-negative combinations of inequalities in λ and $1 \geq 0$ (cf. Corollary 1).

Step 3. Applying Handelman's Theorem and Farkas' Lemma. For every $\gamma = (\lambda, \varrho) \in \Gamma$ and strong polynomial inequality $g \gg 0$ in ϱ, the algorithm performs the following operations:

- Let $\textsc{Monoid}_d(\lambda) = \{h_1, h_2, \ldots, h_u\}$ be the set of all polynomials in $\textsc{Monoid}(\lambda)$ whose degree is at most d. The algorithm symbolically computes $\textsc{Monoid}_d(\lambda)$ and all of it elements.
- The algorithm considers the following three cases, writes constraints that model each of them, and then combines them disjunctively:
  (i) *Writing g as in Equation 12.* The algorithm creates $u+1$ new variables $\widehat{y_0}, \widehat{y_1}, \ldots, \widehat{y_u}$ with the constraints $\widehat{y_0} > 0$ and $\widehat{y_1}, \ldots, \widehat{y_u} \geq 0$, and symbolically computes the equation
  $$g = \widehat{y_0} + \textstyle\sum_{i=1}^{u} \widehat{y_i} \cdot h_i.$$
  Note that both sides of this equation are polynomials of degree d over $\mathbf{V}$. Hence, they are equal iff they agree on the coefficient of every monomial. The algorithm equates the coefficients of corresponding monomials in the LHS and RHS of the equation above, hence obtaining a set of equalities over template variables.
  (ii) *Obtaining $-1 \geq 0$ as a non-negative combination of λ and $1 \geq 0$.*
  (iii) *Obtaining $0 > 0$ as a non-negative combination of λ and $1 \geq 0$.*
  Cases (ii) and (iii) are handled using Farkas' Lemma in the exact same manner as in our previous algorithm (Section 3.1).
- The algorithm adds the resulting constraints to the non-linear constraint system

Example 15. *Consider our running example (Figure 2) together with the templates generated in Example 6. Assume that we aim to synthesize an IRW for $\tau_3 := \left(x^2 - x - 100 \gg 0\right)$, and no target sets in other locations. When Step 2 of the algorithm is applied to location 3 (in exactly the same manner as in Section 3.1) it creates several constraint pairs, including the following:*

$$\lambda : \begin{cases} \widehat{c_{12}} + \widehat{c_{13}} \cdot x + \widehat{c_{14}} \cdot y \geq 0 \\ \widehat{c_{15}} + \widehat{c_{16}} \cdot x + \widehat{c_{17}} \cdot y \geq 0 \\ -\widehat{c_3} - 5 \cdot \widehat{c_4} - \widehat{c_4} \cdot x - \widehat{c_5} \cdot y > 0 \end{cases} \qquad \varrho : (x^2 - x - 100 \gg 0)$$

In Step 3 of the algorithm, the constraint pair $\gamma = (\lambda, \varrho)$ is handled as follows:

- *The algorithm computes $\textsc{Monoid}_2(\lambda)$ which consists of all products of polynomials in λ up to degree 2. Explicitly, it computes an expanded version of the following polynomials:*

$$\begin{array}{ll} h_1 := 1 & h_2 := \widehat{c_{12}} + \widehat{c_{13}} \cdot x + \widehat{c_{14}} \cdot y \\ h_3 := \widehat{c_{15}} + \widehat{c_{16}} \cdot x + \widehat{c_{17}} \cdot y & h_4 := -\widehat{c_3} - 5 \cdot \widehat{c_4} - \widehat{c_4} \cdot x - \widehat{c_5} \cdot y \\ h_5 := h_2^2 & h_6 := h_2 \cdot h_3 \\ h_7 := h_2 \cdot h_4 & h_8 := h_3^2 \\ h_9 := h_3 \cdot h_4 & h_{10} := h_4^2 \end{array}$$

- *The algorithm considers cases (i)-(iii) as above. Cases (ii) and (iii) are similar to Section 3.1, so we focus on (i). The algorithm introduces 11 new variables $\widehat{y_0}, \ldots, \widehat{y_{10}}$, adds the constraints $\widehat{y_0} > 0$ and $\widehat{y_1} \ldots \widehat{y_{10}} \geq 0$ and symbolically computes the following equality:*

$$x^2 - x - 100 = \widehat{y_0} + \sum_{i=1}^{10} \widehat{y_i} \cdot h_i$$

As before, this is a polynomial equality in $\mathbb{R}[x, y]$, and must hold for all values of x, y. So, the corresponding coefficients of the two sides should be equal. The algorithm generates these equalities. For example, given that the constant factor must be the same in the LHS and RHS, the algorithm generates this equality:
$-100 = \widehat{y_0} + \widehat{y_1} + \widehat{y_2} \cdot \widehat{c_{12}} + \widehat{y_3} \cdot \widehat{c_{15}} - \widehat{y_4} \cdot \widehat{c_3} - 5 \cdot \widehat{y_4} \cdot \widehat{c_4} + \widehat{y_5} \cdot \widehat{c_{12}}^2 + \widehat{y_6} \cdot \widehat{c_{12}} \cdot \widehat{c_{15}} - \widehat{y_7} \cdot \widehat{c_{12}} \cdot \widehat{c_3} - 5 \cdot \widehat{y_7} \cdot \widehat{c_{12}} \cdot \widehat{c_4} + \widehat{y_8} \cdot \widehat{c_{15}}^2 - \widehat{y_9} \cdot \widehat{c_{15}} \cdot \widehat{c_3} - 5 \cdot \widehat{y_9} \cdot \widehat{c_{15}} \cdot \widehat{c_4} + \widehat{y_{10}} \cdot \widehat{c_3}^2 + 10 \cdot \widehat{y_{10}} \cdot \widehat{c_3} \cdot \widehat{c_4} + 25 \cdot \widehat{y_{10}} \cdot \widehat{c_4}^2$.
The algorithm generates similar equalities for the coefficients of $x, y, x^2, x \cdot y$, and y^2.

Note that Steps 4 and 5 are also exactly the same as in our previous algorithm and are omitted here. This being said, we have the following theorems, whose proofs are similar to Section 3.1:

Theorem 13 (Soundness). *Given an input $k-$linear system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, and a set τ_ℓ of at most k polynomial inequalities of degree d or less at every $\ell \in \mathbf{L}$, every solution of the non-linear constraint system solved in Step 5 of the algorithm above produces a valid $k-$linear IRW/UIRW for a target set $\mathbf{T}$ that satisfies τ_ℓ at every $\ell \in \mathbf{L}$.*

Theorem 14 (Completeness). *Given a $k-$linear system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, and a set τ_ℓ of at most k strong polynomial inequalities of degree d or less at every $\ell \in \mathbf{L}$, every bounded $k-$linear IRW/UIRW for a target set $\mathbf{T}$ that satisfies τ_ℓ at every $\ell \in \mathbf{L}$, is produced by some solution of the non-linear constraint system solved in Step 5 of the algorithm above.*

Theorem 15 (Complexity). *For fixed constants k, d and β, given a $k-$linear β-branching system $S = (\mathbf{V}, \mathbf{L}, \ell_0, I, \Theta)$, and a set τ_ℓ of at most k polynomial inequalities of degree d or less at every $\ell \in \mathbf{L}$, Steps 1–4 of the algorithm above lead to a* polynomial-time *reduction from the problem of generating a k-linear IRW/UIRW to solving a QP instance.*

Remark 4. *Unlike the linear case, our completeness result in Theorem 14 requires* strong *inequalities and* boundedness. *This is because Handelman's theorem is only applicable when $\textsc{Sat}(\Phi)$ is compact, and hence Corollary 3 can only handle*

strong inequalities over bounded polyhedra. These requirements do not apply to our soundness result, and although they are theoretically necessary, they have very little impact in practice. If there is an IRW/UIRW for a target set $\mathbf{T}$ *that ensures reachability within* n *steps, it is easy to verify that there is also a bounded IRW/UIRW with the same property, i.e. the semi-runs starting at* ν_0 *and taking* n *transitions cannot visit an unbounded set of valuations. Moreover, if the target set contains a non-strong inequality such as* $g \geq 0$ *or* $g > 0$*, one can replace this inequality with* $g + \overline{\epsilon} \gg 0$ *for a new variable* $\overline{\epsilon} \geq 0$ *and solve a quadratic programming instance with the goal of minimizing* $\overline{\epsilon}$*. This trick will slightly change the problem, but it rarely has practical significance.*

F Enforcing a Polynomial to be a Sum of Squares

In several places in our algorithm, we have a sum-of-squares polynomial $\widehat{h}$ defined by a template

$$\widehat{h} := \widehat{\eta_1} \cdot \mathfrak{m}_1 + \ldots + \widehat{\eta_{\mathfrak{n}}} \cdot \mathfrak{m}_{\mathfrak{n}}$$

in which $\{\mathfrak{m}_1, \ldots, \mathfrak{m}_{\mathfrak{n}}\}$ are monomials over a set $\mathbf{V}$ of variables, the $\widehat{\eta_i}$'s are unknown reals, and the algorithm depends on ensuring that $\widehat{h}$ is indeed a sum-of-squares polynomial. Given that our algorithm reduces the problem of generating an IRW/UIRW to quadratic programming, we would like to similarly reduce the problem of $\widehat{h}$ being a sum-of-squares to quadratic programming over the $\widehat{\eta_i}$'s. In this section, we show how such a reduction can be obtained. This is a standard procedure and has previously been used in many other constraint-based program analysis algorithms. The presentation we use is taken from [36]. Our main tools are two well-known theorems:

Theorem 16 ([72, Chapter 7])**.** *A polynomial* $\widehat{h} \in \mathbb{R}[\mathbf{V}]$ *of even degree* $\mathfrak{d}$ *is a sum-of-squares iff there exists an* $\mathfrak{r}-$*dimensional symmetric positive semi-definite matrix* $\mathcal{P}$ *such that* $h = y^T \mathcal{P} y$*, where* $\mathfrak{r}$ *is the number of monomials of degree no greater than* $\mathfrak{d}/2$ *and* y *is a column vector consisting of every such monomial.*

Theorem 17 ([69])**.** *A symmetric square matrix* $\mathcal{P}$ *is positive semi-definite iff it has a Cholesky decomposition of the form* $\mathcal{P} = \mathcal{L}\mathcal{L}^T$ *where* $\mathcal{L}$ *is a lower-triangular matrix with non-negative diagonal entries.*

Given the two theorems above, we use the following standard process for generating quadratic equations that are equivalent to $\widehat{h}$ being a sum-of-squares:

Generating Sum-of-Squares Constraints. The algorithm generates the set $M_{\frac{\mathfrak{d}}{2}}$ consisting of all monomials of degree at most $\frac{\mathfrak{d}}{2}$ over $\mathbf{V}$ and creates a vector y of these monomials. It then symbolically computes the following equality:

$$\widehat{h} = y^T \widehat{\mathcal{L}} \widehat{\mathcal{L}}^T y.$$

Here, $\widehat{\mathcal{L}}$ is a lower-triangular matrix. Every entry of $\widehat{\mathcal{L}}$ is a new unknown variable, and every diagonal entry is constrained to be non-negative. As usual, the algorithm equates the corresponding terms on both sides of this polynomial equality to obtain quadratic equations over the unknown variables. It follows directly from the two theorems above that this reduction is both sound and complete.

Example 16 (Taken from [36])**.** *Let* $\mathbf{V} = \{a, b\}$ *be the set of variables and* $\widehat{h} \in \mathbb{R}[\mathbf{V}]$ *a quadratic polynomial, i.e.* $\widehat{h}(a, b) = \widehat{\eta_1} + \widehat{\eta_2} \cdot a + \widehat{\eta_3} \cdot b + \widehat{\eta_4} \cdot a^2 + \widehat{\eta_5} \cdot a \cdot b + \widehat{\eta_6} \cdot b^2$*. We aim to encode the property that* $\widehat{h}$ *is a sum-of-squares as a system of quadratic equalities and inequalities. To do so, we first generate all monomials of degree at most* $\lfloor \mathfrak{d}/2 \rfloor = 1$*, which are* 1*,* a *and* b*. Hence, we let* $y = \begin{bmatrix} 1 & a & b \end{bmatrix}^T$*. We then generate a template for a lower-triangular matrix* $\widehat{\mathcal{L}}$ *whose every non-zero entry is a new variable:*

$$\widehat{\mathcal{L}} = \begin{bmatrix} \widehat{l_1} & 0 & 0 \\ \widehat{l_2} & \widehat{l_3} & 0 \\ \widehat{l_4} & \widehat{l_5} & \widehat{l_6} \end{bmatrix}.$$

We also add the inequalities $\widehat{l_1} \geq 0, \widehat{l_3} \geq 0$ *and* $\widehat{l_6} \geq 0$ *to our system. Now, we write the equation* $\widehat{h} = y^T \widehat{\mathcal{L}} \widehat{\mathcal{L}}^T y$ *and compute it symbolically:*

$$\widehat{h} = \begin{bmatrix} 1 & a & b \end{bmatrix} \begin{bmatrix} \widehat{l_1} & 0 & 0 \\ \widehat{l_2} & \widehat{l_3} & 0 \\ \widehat{l_4} & \widehat{l_5} & \widehat{l_6} \end{bmatrix} \begin{bmatrix} \widehat{l_1} & \widehat{l_2} & \widehat{l_4} \\ 0 & \widehat{l_3} & \widehat{l_5} \\ 0 & 0 & \widehat{l_6} \end{bmatrix} \begin{bmatrix} 1 \\ a \\ b \end{bmatrix},$$

which leads to:

$$\widehat{\eta_1} + \widehat{\eta_2} \cdot a + \widehat{\eta_3} \cdot b + \widehat{\eta_4} \cdot a^2 + \widehat{\eta_5} \cdot a \cdot b + \widehat{\eta_6} \cdot b^2 =$$
$$\widehat{l_1}^2 + 2 \cdot \widehat{l_1} \cdot \widehat{l_2} \cdot a + 2 \cdot \widehat{l_1} \cdot \widehat{l_4} \cdot b + (\widehat{l_2}^2 + \widehat{l_3}^2) \cdot a^2$$
$$+ (2 \cdot \widehat{l_2} \cdot \widehat{l_4} + 2 \cdot \widehat{l_3} \cdot \widehat{l_5}) \cdot a \cdot b + (\widehat{l_4}^2 + \widehat{l_5}^2 + \widehat{l_6}^2) \cdot b^2.$$

Note that both sides of the equation above are polynomials over $\{a, b\}$*, hence they are equal iff their corresponding coefficients are equal. So, we get the following quadratic equalities over the* $\widehat{\eta_i}$*'s and* $\widehat{l_i}$*'s:* $\widehat{\eta_1} = \widehat{l_1}^2, \widehat{\eta_2} = 2 \cdot \widehat{l_1} \cdot \widehat{l_2}, \ldots, \widehat{\eta_6} = \widehat{l_4}^2 + \widehat{l_5}^2 + \widehat{l_6}^2$*. This concludes the construction of our quadratic system.*

G Polynomial Programs used in the Experimental Results

In this section, we provide details of the polynomial programs that were used in our experiments in Section 4. Figures 4–9 show the programs. We now discuss each benchmark in more detail:

- `sqrt1`: This program gets a value n as input and computes $\lfloor \sqrt{n} \rfloor$ by simply iterating through every integer starting from 1. The goal is to reach the end of the program with $n - s > 10^5$. Therefore, to solve this task, a verifier has to assign a value to n such that $n - \lfloor \sqrt{n} \rfloor > 10^5$. It is easy to see that any $n > 10^5 + 316$ works. However, this

example is interesting because the shortest path to a target state needs to go through 316 iterations of the **while** loop. Moreover, the loop has a quadratic guard. As such, a verifier that is based on abstract interpretation needs to obtain a relatively fine abstraction, whereas approaches based on loop-unrolling and symbolic execution need to unroll this non-linear loop 316 times. As mentioned in Table 2, CPAchecker times out on this instance. However, VeriAbs succeeds in proving reachability in 207.3 seconds. In contrast, our approach synthesizes an IRW in just 40.8 seconds.

$$
\begin{aligned}
&I : n \geq 1 \ \wedge \ s = 1\\
&\textbf{while} \ \ (s+1)^2 \leq n :\\
&\qquad s := s + 1
\end{aligned}
$$

Figure 4. The program sqrt1. Our target is to reach the end of this program with $n - s > 10^5$.

- sqrt2: This is a variant of sqrt1 in which the value of s is doubled in every step if $2 \cdot s \leq \sqrt{n}$. This simple change means that there are now many short paths that reach the target. For example, by setting $n = 2^{18}$, one can reach the target in just 9 iterations. Unsurprisingly, both CPAchecker and VeriAbs can handle this example (Table 2). This being said, note that the complexity of our approach does not depend on the length of paths. As such, while sqrt1 is much harder than sqrt2 for other approaches, our runtimes on these two benchmarks are very close. Indeed, our method solves sqrt1 a bit faster than sqrt2 (40.8s vs 46.4s). This is because sqrt1 is a smaller program.

$$
\begin{aligned}
&I : n \geq 1 \ \wedge \ s = 1\\
&\textbf{while} \ \ (s+1)^2 \leq n :\\
&\qquad \textbf{if} \ \ 4 \cdot s^2 \leq n :\\
&\qquad\qquad s := 2 \cdot s\\
&\qquad \textbf{else} :\\
&\qquad\qquad s := s + 1
\end{aligned}
$$

Figure 5. The program sqrt2. Our target is to reach the end of this program with $n - s > 10^5$.

- sum: This program simply sums up all the integers from 1 to a given value n. Note that the program itself is linear (the loop guard and the updates are linear). However, we need polynomial arguments given that for every integer n, at the end of this program we will have $s = \frac{n \cdot (n+1)}{2}$. As in the previous examples, the target set can only be reached after many iterations. To reach the target, it suffices to choose $10000 \leq n \leq 11000$. Our algorithm is exact and can handle tight inequalities. We chose this liberal interval in order to make the instance solvable for abstract interpretation approaches with good precision. Nevertheless, CPAchecker timed out and VeriAbs terminated with no result, i.e. returned "unknown".

$$
\begin{aligned}
&I : s = 0 \ \wedge \ i = 1\\
&\textbf{while} \ \ i \leq n :\\
&\qquad (s, i) := (s + i, i + 1)
\end{aligned}
$$

Figure 6. The program sum. Our target is to reach the end of this program with $50005000 \leq s \leq 60505500$.

- sum2: This program is similar to sum but it adds the squares of integers from 1 to n. Because this program has non-linear assignments, it is intuitively harder to verify in comparison with sum.

$$
\begin{aligned}
&I : s = 0 \ \wedge \ i = 1\\
&\textbf{while} \ \ i \leq n :\\
&\qquad (s, i) := (s + i^2, i + 1)
\end{aligned}
$$

Figure 7. The program sum2. Our target is to reach the end of this program with $333383335000 \leq s \leq 443727168500$.

- robot1: This program models the behavior of two robots on a 2d plane. One robot is located at (x_1, y_1) and the other at (x_2, y_2). Initially, we have $(x_1, y_1) = (x_2, y_2)$. At each iteration, each robot moves one unit upwards or to the right. The direction is chosen non-deterministically. The goal is to prove reachability to the endpoint of the program. This is equivalent to proving that it is possible for the robots to move in such a way that makes their distance from each other more than $\sqrt{10^5}$. The main difficulty in this program is the combinatorial explosion in the number of paths. Nevertheless, note that a relatively large proportion of the paths lead to the desired target. As such, it was surprising for us to see that our approach was the only one that succeeded in handling this example.

$$
\begin{aligned}
&I : x_1 = x_2 \ \wedge \ y_1 = y_2\\
&\textbf{while} \ \ (x_1 - x_2)^2 + (y_1 - y_2)^2 \leq 10^5 :\\
&\qquad x_1 := x_1 + 1 \ \square \ y_1 := y_1 + 1\\
&\qquad x_2 := x_2 + 1 \ \square \ y_2 := y_2 + 1
\end{aligned}
$$

Figure 8. The program robot1. Our target is to reach the end of this program.

- robot2: This is a variant of robot1 which intuitively seems to be a bit harder. The same two robots are put on opposite sides of a square with side-length 10^4 and the goal is to prove that they can move in a way that decreases their distance to less than 10. This example creates the same combinatorial explosion in the number of possible paths as in robot1, but this time, only a very small proportion of these paths reach the target. Nevertheless, our

approach can handle this example in virtually the same amount of time as `robot1`. This is because our approach is (semi-)complete and finds a polynomial IRW if one exists. It does not depend on the proportion of paths that lead to a target state.

$$
\begin{aligned}
&I : y_1 = y_2 + 10^4 \ \wedge \ x_1 = x_2 - 10^4\\
&\textbf{while} \ \ (x_1 - x_2)^2 + (y_1 - y_2)^2 \geq 100 :\\
&\qquad x_1 := x_1 + 1 \ \square \ y_1 := y_1 + 1\\
&\qquad x_2 := x_2 + 1 \ \square \ y_2 := y_2 + 1
\end{aligned}
$$

Figure 9. The program `robot2`. Our target is to reach the end of this program.

H Related Works

Below, we compare our approach with several families of previous results.

CEGAR-based Model-Checkers. Counterexample-guided abstraction refinement [1, 7, 14, 38] is one of the most successful ideas in software verification and has been implemented in many model-checkers, including the well-known tools SLAM [16] and BLAST [24], which handle not only safety properties, but also problems such as test-case generation [23]. These model-checkers repeatedly run reachability analyses in order to obtain the required counterexamples for refining their abstractions. A significant challenge is that when variable domains are infinite or uncountable, e.g. $\mathbb{R}$, they cannot guarantee both termination and completeness. They either provide a complete approach that might not terminate, or a sound terminating approach with no completeness guarantee. Another significant challenge arises when there are many spurious counterexamples. Approaches for mitigating this problem also rely on reachability (e.g. see [20]).

Invariant Generation. Invariant generation aims to automatically generate over-approximations of reachable sets, while our approach targets reachability analysis that aims to check whether certain undesirable states can be reached (whether a bug is present in the program). Note that although we have the same inductiveness idea as in inductive invariant generation, the idea leads to *under-approximation* (rather than over-approximation) of the set of states that can reach some target state. Invariant generation is very well-studied and many approaches are developed for automating it, including recurrence analysis [54, 75, 76], abstract interpretation [11, 32, 55, 59], constraint-solving [36, 40], inference [97] and symbolic execution [45].

Symbolic Execution. Symbolic execution [28, 29] runs program codes in a static and symbolic manner, thus it is very effective for analyzing programs without loops or with bounded loops. When tackling loops, symbolic execution can only unfold the loop a bounded number of steps, hence it is unsuitable for loops with an unbounded number of iterations. In contrast, our approach can handle loops with unbounded iterations, and provides soundness and completeness.

Abstract Interpretation. Abstract interpretation mainly focuses on over-approximation of reachable states through the widening operator, which often leads to a loss of precision [64]. There are also several abstraction-based results on under-approximation [61, 91, 93, 96]. However, a theory with completeness guarantees for generating under-approximations such as our $\mathbf{T}$-inductive sets through abstract interpretation is still lacking.

Termination Analysis. Termination analysis only considers whether a program terminates in a finite number of steps, i.e. whether the program can reach the terminal program counter or not, without constraints over program variables. In our approach, we consider reachability to program states defined by numerical constraints over program variables, which is a considerable extension of the termination property. The primary method of proving termination is to synthesize a ranking function, and there are template-based algorithms for the synthesis of linear/polynomial ranking functions [34, 41, 78, 88]. Termination and reachability properties have also been extensively studied in the context of probabilistic programs, especially through martingale-based approaches (e.g. see [31, 73, 100]).

Incorrectness Logic. Incorrectness logic [86] and reverse Hoare logic [50] are similar to Hoare logic but target under-approximations. The logical background behind our approach is a bit different from incorrectness logic. Incorrectness logic obtains under-approximations of the set of reachable states. Hence, a bug can be found by taking the intersection of the under-approximation obtained by incorrectness logic and the set $\mathbf{T}$ of undesirable states. In contrast, we find under-approximations of the sets of states from which reachability to an undesirable state (or bug) in $\mathbf{T}$ is guaranteed. Intuitively, the relationship between our approach and incorrectness logic is similar to the relationship between inductive invariants and Hoare logic. It is also noteworthy that incorrectness logic needs manual effort to write assertions, while our approach is entirely automated when we consider linear/polynomial IRWs/UIRWs.