Verification Constraint Problems with Strengthening

Aaron R. Bradley and Zohar Manna

Computer Science Department
Stanford University
Stanford, CA 94305-9045
{arbrad,manna}@cs.stanford.edu

Abstract. The deductive method reduces verification of safety properties of programs to, first, proposing inductive assertions and, second, proving the validity of the resulting set of first-order verification conditions. We discuss the transition from *verification conditions* to *verification constraints* that occurs when the deductive method is applied to parameterized assertions instead of fixed expressions (*e.g.*, $p_0 + p_1 j + p_2 k \geq 0$, for parameters p_0, p_1, and p_2, instead of $3 + j - k \geq 0$) in order to discover inductive assertions. We then introduce two new verification constraint forms that enable the incremental and property-directed construction of inductive assertions. We describe an iterative method for solving the resulting constraint problems. The main advantage of this approach is that it uses off-the-shelf constraint solvers and thus directly benefits from progress in constraint solving.

1 Introduction

The deductive method of program verification reduces the verification of *safety* and *progress* properties to proving the validity of a set of first-order *verification conditions* [13]. In the safety case, the verification conditions assert that the given property is *inductive*: it holds initially (initiation), and it is preserved by taking any transition (consecution). Such an assertion is an *invariant* of the program. In the progress case, the verification conditions assert that a given function is bounded from below (bounded), yet decreases when any transition is taken (ranking). The existence of such a function, called a *ranking function*, guarantees that the program terminates. In this paper, we focus on the generation of inductive assertions. Section 5 briefly discusses the application of similar techniques to the synthesis of ranking functions.

We discuss a natural shift in perspective from verification conditions to *verification constraints*. This shift is achieved by replacing the given assertion with a *parameterized* assertion. The task is then to find some instantiation of the parameters that validates the verification conditions. This task is a constraint satisfaction problem. Now, each verification condition is a constraint on the parameters. Instantiating the parameters of the parameterized assertion with a solution produces an inductive assertion. This method of generating inductive invariants is known as the *constraint-based* method [7].

In general, these verification constraint problems (VCPs) have many solutions, only some of which are interesting. In particular, a solution is only interesting if it provides more information than what is already known. If χ_i is the currently known inductive invariant and φ is a new solution to a safety VCP, then $\chi_{i+1} \stackrel{\text{def}}{=} \chi_i \wedge \varphi$ should be stronger: χ_{i+1} implies χ_i, but χ_i should not imply χ_{i+1}. We introduce the new verification constraint form called strengthening to ensure this property.

Additionally, we could have a safety property Π in mind when analyzing a program. A safety property is an assertion that holds on all reachable states of the program. In this context, solutions that strengthen Π are sometimes more interesting than solutions that simply strengthen the currently known invariant χ_i. We introduce the new verification constraint form called Π-strengthening to facilitate property-directed invariant generation.

We present basic concepts in Section 2. In Section 3, we introduce verification constraints as a natural generalization of verification conditions through a set of examples. We also introduce the two new forms of verification constraints that enable an incremental construction of invariants. Section 4 then turns to the task of solving the verification constraint problems. To address the existential constraints arising from the new forms of verification constraints, we describe an iterative method of constructing incrementally stronger inductive assertions using general constraint solvers. Section 5 then reviews past work on posing and solving verification constraint problems in areas ranging from program analysis to continuous and hybrid systems. Section 6 concludes.

2 Preliminaries

It is standard practice to formalize programs as mathematical objects called transition systems. For our purposes in this paper, we define a simple form of transition system.

Definition 1 (Transition System) A *transition system* $\mathcal{S} \colon \langle \overline{x}, \theta, \mathcal{T} \rangle$ contains three components:

- a set of *variables* $\overline{x} = \{x_1, \ldots, x_n\}$ that range over the integers $\mathbb{Z}$ or reals $\mathbb{R}$;
- an assertion $\theta[\overline{x}]$ over $\overline{x}$ specifying the *initial condition*;
- and a set of transitions $\mathcal{T} = \{\tau_1, \ldots, \tau_k\}$, where each transition τ is specified by its *transition relation* $\rho_\tau[\overline{x}, \overline{x}']$, an assertion over $\overline{x}$ and $\overline{x}'$.

Primed variables $\overline{x}'$ represent the next-state values of their corresponding variables $\overline{x}$.

Thus, in this paper, a transition system consists simply of a set of transitions that loop around a single program point. Variables have integer or real type. Finally, the computation model is sequential, as formalized next.

Definition 2 (State & Computation) A *state* s of a transition system $\mathcal{S}$ is an assignment of values to its variables. A *computation* $\sigma\colon s_0, s_1, s_2, \ldots$ is an infinite sequence of states such that

- s_0 is an initial state: $s_0 \models \theta$;
- for each $i \geq 0$, each adjacent pair of states is related by some transition: $\exists \tau \in \mathcal{T}.\ (s_i, s_{i+1}) \models \rho_\tau$.

Definition 3 (Invariant) An assertion φ is an *invariant* of $\mathcal{S}$, or is $\mathcal{S}$-*invariant*, if for all computations $\sigma\colon s_0, s_1, s_2, \ldots$, for all $i \geq 0$, $s_i \models \varphi$. An assertion φ is $\mathcal{S}$-*inductive* if

- it holds initially: $\forall \overline{x}.\ \theta[\overline{x}] \rightarrow \varphi[\overline{x}]$; (initiation)
- it is preserved by every $\tau \in \mathcal{T}$: $\forall \overline{x}, \overline{x}'.\ \varphi[\overline{x}] \wedge \rho_\tau[\overline{x}, \overline{x}'] \rightarrow \varphi[\overline{x}']$. (consecution)

If φ is $\mathcal{S}$-inductive, then it is $\mathcal{S}$-invariant.

For convenience, we employ the following abbreviation: $\varphi \Rightarrow \psi$ abbreviates $\forall \overline{x}.\ \varphi \rightarrow \psi$; $\Rightarrow$ is the logical entailment operator. Then, for example, initiation and consecution are expressed as $\theta \Rightarrow \varphi$ and $\varphi \wedge \rho_\tau \Rightarrow \varphi'$, respectively.

A *safety property* $\Box \Pi$ asserts that a transition system $\mathcal{S}$ does not do anything bad: it never reaches a $\neg\Pi$-state. In other words, Π is $\mathcal{S}$-invariant. Proving safety properties in practice typically consists of finding a *strengthening* inductive assertion. That is, to prove that Π is $\mathcal{S}$-invariant, find an $\mathcal{S}$-inductive assertion φ that entails Π: $\varphi \Rightarrow \Pi$.

Recall that a computation was defined to be an *infinite* sequence of states. Thus, if $\mathcal{S}$ terminates, it does not have any computations according to the current definition of transition systems. Let us agree that every transition system $\mathcal{S}$ has an extra *idling* transition τ_{idle} that does not modify any of $\overline{x}$. Then if $\mathcal{S}$ terminates, only this transition is taken. Thus, the computations of a terminating $\mathcal{S}$ all have infinite τ_{idle}-suffixes.

The first-order formulae that arise according to the initiation and consecution conditions of an assertion and program are called *verification conditions*.

3 Verification Constraint Problems with Strengthening

Having formalized our computation model, we now turn to the main topic: *verification constraints* and *verification constraint problems* (VCPs). In the last section, we saw that *verification conditions* are imposed on a given assertion φ. Suppose instead that φ is a *parameterized assertion*.

Definition 4 (Parameterized Assertion) A *parameterized assertion* over variables $\overline{x}$ has the form

$$p_0 + p_1 t_1 + \cdots + p_m t_m \geq 0\ ,$$

where t_i are monomials over $\overline{x}$ (*e.g.*, x_1, x_1^2, $x_1 x_4^3$), and $\overline{p}$ are parameters.

```
int j, k;
@ j = 2 ∧ k = 0
while (···) do
  if (···)
  then j := j + 4;
  else j := j + 2;
       k := k + 1;
done
```

(a) Simple

```
int x1, x2, y1, y2, y3, y4;
@ (y1 = x1 ∧ y2 = x2 ∧
   y3 = x2 ∧ y4 = 0)
while (y1 ≠ y2) do
  if (y1 > y2)
  then y1 := y1 − y2;
       y4 := y4 + y3;
  else y2 := y2 − y1;
       y3 := y3 + y4;
done
```

(b) GCD-LCM

```
int u, w, x, z;
@ x ≥ 1 ∧ u = 1 ∧ w = 1 ∧ z = 0
while (w ≤ x) do
  (z, u, w) := (z + 1, u + 2, w + u + 2);
done
```

(c) Sqrt

Fig. 1. Example functions

For a given set of parameters that range over some domain, a parameterized assertion represents a set of assertions — one assertion for each instantiation of the parameters. Then the verification conditions become constraints over the parameters. The constraint problem is to find an instantiation of the parameters such that the verification conditions are valid. Usually, the parameters are considered to range over the rationals, although considering only the integers is sufficient since only the ratio between parameters matters, not their absolute values.

In this section, we examine a set of examples illustrating VCPs and their solutions. The first two examples perform invariant generation to discover information about the loops. The strengthening condition guides the solver to discover new information on each iteration of constraint solving. The final example illustrates property-directed invariant generation, in which the Π-strengthening condition guides the solver to discover inductive assertions that eliminate error states.

Example 1 (Simple). Consider the loop Simple in Figure 1**(a)**, which first appeared in [11]. The corresponding transition system $\mathcal{S}$ is the following:

$$\begin{aligned} \overline{x}&: \ \{j,k\} \\ \theta&: \ j = 2 \ \wedge \ k = 0 \\ \rho_{\tau_1}&: \ j' = j + 4 \ \wedge \ k' = k \\ \rho_{\tau_2}&: \ j' = j + 2 \ \wedge \ k' = k + 1 \end{aligned}$$

Because the guards are replaced by nondeterministic choice ($\cdots$ in Figure 1**(a)**), the transitions τ_1 and τ_2 are always enabled.

To prove that the assertion $k \geq 0$ is $\mathcal{S}$-inductive requires checking the validity of the following verification conditions:

- $j = 2 \;\wedge\; k = 0 \;\Rightarrow\; k \geq 0$ (initiation)
- $k \geq 0 \;\wedge\; j' = j + 4 \;\wedge\; k' = k \;\Rightarrow\; k' \geq 0$ (consecution 1)
- $k \geq 0 \;\wedge\; j' = j + 2 \;\wedge\; k' = k + 1 \;\Rightarrow\; k' \geq 0$ (consecution 2)

Simplifying according to equations yields:

- $0 \geq 0$ (initiation)
- $k \geq 0 \;\Rightarrow\; k \geq 0$ (consecution 1)
- $k \geq 0 \;\Rightarrow\; k + 1 \geq 0$ (consecution 2)

These verification conditions are clearly valid.

Now suppose that we want to discover facts about the loop in the form of a conjunction of affine assertions that is $\mathcal{S}$-invariant. An affine expression is a linear combination of variables with an additional constant; an affine assertion asserts that an affine expression is nonnegative. Our strategy is to construct incrementally an $\mathcal{S}$-inductive assertion χ by solving a sequence of VCPs.

Suppose that χ_{i-1} has been discovered so far, where $\chi_0 \stackrel{\text{def}}{=} \mathsf{true}$. On the ith iteration, construct the parameterized assertion

$$\underbrace{p_0 + p_1 j + p_2 k}_{I[j,k]} \geq 0 \;,$$

and solve the following VCP:

$$\begin{array}{ll} \quad I[2,0] \geq 0 & \text{(initiation)} \\ \wedge\;\; \chi_{i-1} \;\wedge\; I \geq 0 \;\Rightarrow\; I[j+4,k] \geq 0 & \text{(consecution 1)} \\ \wedge\;\; \chi_{i-1} \;\wedge\; I \geq 0 \;\Rightarrow\; I[j+2,k+1] \geq 0 & \text{(consecution 2)} \\ \wedge\;\; \exists j,k.\; \chi_{i-1} \;\wedge\; I < 0 & \text{(strengthening)} \end{array}$$

The notation $I[j+4,k] \geq 0$ simplifies $j' = j + 4 \;\wedge\; k' = k \;\Rightarrow\; I[j',k'] \geq 0$. The first conjunct imposes initiation; the next two conjuncts impose consecution for each of the two paths of the loop. Together, these three conjuncts constrain $I \geq 0$ to be inductive relative to χ_{i-1}. The final conjunct asserts that there is some χ_{i-1}-state s that is excluded by the new invariant. We call this constraint the strengthening condition.

Any solution for (p_0, p_1, p_2) of this VCP represents an inductive assertion φ_i that strengthens χ_{i-1}. Set $\chi_i \stackrel{\text{def}}{=} \chi_{i-1} \;\wedge\; \varphi_i$, and generate the next VCP with χ_i instead of χ_{i-1}. If a solution does not exist, then halt the incremental construction.

Section 4 discusses how to solve the sequence of VCPs. For now, one possible sequence of discovered inductive assertions is the following:

$$\varphi_1 :\; k \geq 0 \qquad \varphi_2 :\; j \geq 0 \qquad \varphi_3 :\; j \geq 2k + 2 \;,$$

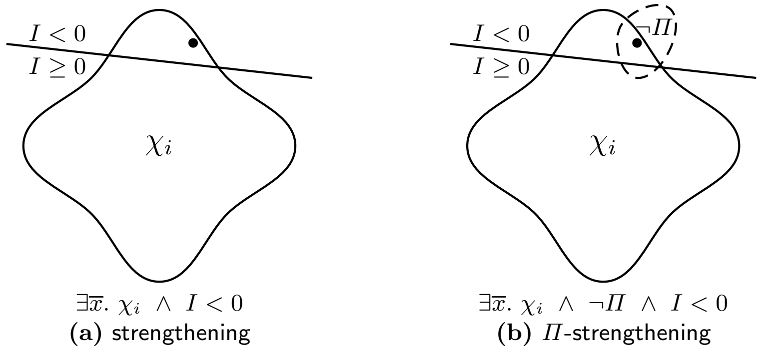

Fig. 2. Illustration of the two strengthening conditions

corresponding to the following sequence of χ_i's:

$$\begin{array}{lllll}
\chi_0 & \stackrel{\text{def}}{=} & \mathsf{true} & \Rightarrow & \chi_0 : \ \mathsf{true} \\
\chi_1 & \stackrel{\text{def}}{=} & \chi_0 \ \wedge \ \varphi_1 & \Rightarrow & \chi_1 : \ k \geq 0 \\
\chi_2 & \stackrel{\text{def}}{=} & \chi_1 \ \wedge \ \varphi_2 & \Rightarrow & \chi_2 : \ k \geq 0 \ \wedge \ j \geq 0 \\
\chi_3 & \stackrel{\text{def}}{=} & \chi_2 \ \wedge \ \varphi_3 & \Rightarrow & \chi_3 : \ k \geq 0 \ \wedge \ j \geq 0 \ \wedge \ j \geq 2k+2
\end{array}$$

Conjuncts $k \geq 0$ and $j \geq 2k + 2$ entail $j \geq 2$. Therefore, χ_3 as well as $j \geq 2$ are $\mathcal{S}$-invariant. Moreover, every χ_i and $j \geq 2$ are $\mathcal{S}$-inductive.

The strengthening condition is essential for making progress. For example, the sequence of inductive assertions

$$\varphi_1 : \ k \geq 0 \qquad \varphi_2 : \ k \geq 0 \qquad \varphi_3 : k \geq 0 \qquad \ldots$$

is a solution to the sequence of VCPs constructed without the strengthening condition, but not to the sequence of VCPs constructed with the strengthening condition.

Figure 2**(a)** illustrates the strengthening condition. The new invariant consists of all states at and below the line. This invariant satisfies the strengthening condition: there exists some χ_i-state (the dot) that the invariant excludes.

In the next example, we look at nonlinear properties.

Example 2 (GCD-LCM). The loop GCD-LCM in Figure 1**(b)** computes the greatest common divisor (y_1 and y_2) and the least common multiple ($y_3 + y_4$) of (x_1, x_2). As in Example 1, our goal is to discover information about the loop. In this case, our strategy is to construct an $\mathcal{S}$-inductive conjunction of quadratic polynomial inequalities. Therefore, in each iteration, we use the parameterized assertion

$$\underbrace{\mathsf{Quadratic}(x_1, x_2, y_1, y_2, y_3, y_4)}_{I[\overline{x}, \overline{y}]} \geq 0 \ .$$

$\mathsf{Quadratic}$ forms the most general parameterized quadratic expression over the given variables; *e.g.*,

$$\mathsf{Quadratic}(x, y) = p_0 + p_1 x + p_2 y + p_3 xy + p_4 x^2 + p_5 y^2 \ .$$

In this case, $I[\overline{x}, \overline{y}]$ contains 28 parameterized monomials.

Suppose that χ_{i-1} has been constructed so far. On the ith iteration, solve the following VCP:

$$\begin{array}{ll}
\quad \forall \overline{x}.\ I[x_1, x_2, x_1, x_2, x_2, 0] \geq 0 & (\mathsf{initiation}) \\
\wedge\ \chi_{i-1}\ \wedge\ I \geq 0\ \wedge\ y_1 > y_2\ \Rightarrow\ I[x_1, x_2, y_1 - y_2, y_2, y_3, y_4 + y_3] \geq 0 & \\
\wedge\ \chi_{i-1}\ \wedge\ I \geq 0\ \wedge\ y_1 < y_2\ \Rightarrow\ I[x_1, x_2, y_1, y_2 - y_1, y_3 + y_4, y_4] \geq 0 & \\
 & (\mathsf{consecution}) \\
\wedge\ \exists \overline{x}, \overline{y}.\ \chi_{i-1}\ \wedge\ I < 0 & (\mathsf{strengthening})
\end{array}$$

The first conjunct imposes $\mathsf{initiation}$; the next two impose $\mathsf{consecution}$; and the final conjunct imposes the $\mathsf{strengthening}$ condition, expressing that the new assertion strengthens χ_{i-1}.

One possible sequence of inductive assertion discovery is the following:

$$\begin{array}{lcl}
\chi_0 & \stackrel{\text{def}}{=} & \mathsf{true} \\
\chi_1 & \stackrel{\text{def}}{=} & \chi_0\ \wedge\ \varphi_1 :\ x_1 x_2 - y_1 y_3 - y_2 y_4 \geq 0 \\
\chi_2 & \stackrel{\text{def}}{=} & \chi_1\ \wedge\ \varphi_2 :\ x_1 x_2 - y_1 y_3 - y_2 y_4 \geq 0\ \wedge\ -x_1 x_2 + y_1 y_3 + y_2 y_4 \geq 0
\end{array}$$

Thus, χ_2 implies that

$$x_1 x_2 = y_1 y_3 + y_2 y_4$$

is $\mathcal{S}$-invariant.

In many cases, we have a safety property in mind that we would like to prove. The next example describes such a case. We introduce a variant of the $\mathsf{strengthening}$ condition to direct the invariant generation toward proving the property.

Example 3 (Integer Square-Root). The loop SQRT in Figure 1**(c)** computes the integer square-root z of a positive integer x. On exit, the following relation should hold between z and x:

$$z^2 \leq x < (z+1)^2 \ .$$

Taking preconditions reveals that the following should be invariant at the top of the loop:

$$\Pi :\ (w \leq x\ \rightarrow\ (z+1)^2 \leq x)\ \wedge\ (w > x\ \rightarrow\ x < (z+1)^2) \ .$$

Π is not $\mathcal{S}$-inductive. The goal is to prove that Π is $\mathcal{S}$-invariant by generating a strengthening inductive assertion χ such that $\chi\ \Rightarrow\ \Pi$. As in Example 2, our strategy is to construct a conjunction of at most quadratic polynomial inequalities. Unlike in previous examples, however, our goal is not to discover properties of the loop; rather, it is to prove $\Box \Pi$.

On each iteration, we use the parameterized assertion

$$\underbrace{\mathsf{Quadratic}(u, w, x, z)}_{I[u,w,x,z]} \geq 0 \ .$$

Suppose that χ_{i-1} has been generated so far; on the ith iteration, solve the following VCP:

$$\begin{array}{ll} \quad x \geq 0 \ \Rightarrow \ I[1, 1, x, 0] \geq 0 & \text{(initiation)} \\ \wedge \ \chi_{i-1} \ \wedge \ I \geq 0 \ \wedge \ w \leq x \ \Rightarrow \ I[u+2, w+u+2, x, z+1] \geq 0 & \\ & \text{(consecution)} \\ \wedge \ \exists u, w, x, z. \ \chi_{i-1} \ \wedge \ \neg \Pi \ \wedge \ I < 0 & (\Pi\text{-strengthening}) \end{array}$$

The first two conjuncts express initiation and consecution, respectively. The final conjunct is a variant of strengthening called Π-strengthening: the new inductive assertion should exclude some state that is both a χ_{i-1}-state (so that χ_{i-1} is strengthened) and a $\neg\Pi$-state (so that an error state is excluded, and thus Π is strengthened).

It is not always possible to exclude a $\neg\Pi$-state with an inductive assertion of a fixed form, so some iterations should use the weaker strengthening condition instead.

One run of this incremental construction produces the following assertions, listed in order from the discovered inductive assertion of the first iteration (top-left) to that of the final iteration (bottom-right).

$$\begin{array}{lrlr} \varphi_1 : & -x + ux - 2xz \geq 0 & \varphi_7 : & -1 + u \geq 0 \\ \varphi_2 : & u \geq 0 & \varphi_8 : & -2u - u^2 + 4w \geq 0 \\ \varphi_3 : & u - u^2 + 4uz - 4z^2 \geq 0 & \varphi_9 : & -3 - u^2 + 4w \geq 0 \\ \varphi_4 : & 3u + u^2 - 4w \geq 0 & \varphi_{10} : & -5u - u^2 + 6w \geq 0 \\ \varphi_5 : & x - ux + 2xz \geq 0 & \varphi_{11} : & -15 + 22u - 11u^2 + 4uw \geq 0 \\ \varphi_6 : & 1 + 2u + u^2 - 4w \geq 0 & \varphi_{12} : & -1 - 2u - u^2 + 4w \geq 0 \end{array}$$

Thus, $\chi_{12} \overset{\text{def}}{=} \varphi_1 \ \wedge \ \cdots \ \wedge \ \varphi_{12}$. Each assertion is inductive relative to the previous assertions.

On the thirteenth iteration, it is discovered that no $(\chi_{12} \ \wedge \ \neg\Pi)$-state exists, proving $\Box\Pi$. Specifically, φ_1 and φ_5 entail $u = 1 + 2z$, while φ_6 and φ_{12} entail $4w = (u+1)^2$. Thus, $w = (z+1)^2$, entailing Π.

In the next section, we discuss how the new assertion of each iteration is actually found.

Figure 2**(b)** illustrates the Π-strengthening condition. The new invariant consists of all states at and below the line. This invariant satisfies the Π-strengthening condition: there exists some $(\chi_i \ \wedge \ \neg\Pi)$-state (the dot) that the invariant excludes. Therefore, this new invariant strengthens both χ_i and Π, making progress toward proving $\Box\Pi$.

4 Solving VCPs with Strengthening

Section 5 discusses previous work on solving specific forms of VCPs. In this section, we discuss a refinement of the techniques introduced in [7] and [10]. The method of [7] is complete for linear VCPs, while the method of [10] is sound and efficient but incomplete for polynomial VCPs [10]. In particular, we show how to solve VCPs with a $\mathsf{strengthening}$ or Π-$\mathsf{strengthening}$ condition, which have not been studied before. Solving constraints with such conditions enables a simple incremental construction of invariants, as illustrated in the examples of the previous section.

4.1 Farkas's Lemma

To begin, we review the constraint-solving method based on *Farkas's Lemma* [7] or *Lagrangian relaxation* [10]. Farkas's Lemma relates a constraint system over the *primal* variables (the variables of the transition system $\mathcal{S}$) to a *dual* constraint system over the parameters [23]. It is restricted to affine constraints, as in Example 1.

Theorem 1 (Farkas's Lemma). *Consider the following universal constraint system of affine inequalities over real variables $\overline{x} = \{x_1, \ldots, x_m\}$:*

$$S : \begin{bmatrix} a_{1,0} + a_{1,1}x_1 + \cdots + a_{1,m}x_m \geq 0 \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ a_{n,0} + a_{n,1}x_1 + \cdots + a_{n,m}x_m \geq 0 \end{bmatrix}$$

If S is satisfiable, it entails affine inequality $c_0 + c_1x_1 + \cdots + c_mx_m \geq 0$ iff there exist real numbers $\lambda_1, \ldots, \lambda_n \geq 0$ such that

$$c_1 = \sum_{i=1}^{n} \lambda_i a_{i,1} \quad \cdots \quad c_m = \sum_{i=1}^{n} \lambda_i a_{i,m} \quad c_0 \geq \left(\sum_{i=1}^{n} \lambda_i a_{i,0} \right) .$$

Example 4 (Simple). Consider the VCP developed in Example 1 for Simple of Figure 1**(a)**. Using the tabular notation, the first three conjuncts have this form:

$$\begin{array}{c} \textsf{(initiation)} \\ \\ \hline p_0 + 2p_1 + 0p_2 \geq 0 \end{array}$$

$$\begin{array}{l} \qquad\textsf{(consecution 1)} \\ \chi_i \\ p_0 \qquad\qquad +p_1 j \ +p_2 k \geq 0 \\ \hline (p_0 + 4p_1) \ +p_1 j \ +p_2 k \geq 0 \end{array} \qquad \begin{array}{l} \qquad\textsf{(consecution 2)} \\ \chi_i \\ p_0 \qquad\qquad\qquad +p_1 j \ +p_2 k \geq 0 \\ \hline (p_0 + 2p_1 + p_2) \ +p_1 j \ +p_2 k \geq 0 \end{array}$$

Suppose that this is the first iteration so that $\chi_0 \stackrel{\text{def}}{=} \mathsf{true}$. Then dualizing according to the lemma produces the following set of existential constraints over the parameters and λ-multipliers $\{p_0, p_1, p_2, \lambda_1, \lambda_2\}$:

$$\begin{array}{ll} p_0 + 2p_1 \geq 0 \ \wedge & \text{(initiation)} \\ p_0 + 4p_1 \geq \lambda_1 p_0 \ \wedge \ p_1 = \lambda_1 p_1 \ \wedge \ p_2 = \lambda_1 p_2 \ \wedge & \text{(consecution 1)} \\ p_0 + 2p_1 + p_2 \geq \lambda_2 p_0 \ \wedge \ p_1 = \lambda_2 p_1 \ \wedge \ p_2 = \lambda_2 p_2 & \text{(consecution 2)} \end{array}$$

Clearly, $\lambda_1 = \lambda_2 = 1$, so the constraints are equivalent to

$$p_0 + 2p_1 \geq 0 \ \wedge \ p_1 \geq 0 \ \wedge \ 2p_1 + p_2 \geq 0 \ ,$$

for which solutions for (p_0, p_1, p_2) include $(0, 0, 1)$, $(0, 1, 0)$, and $(-2, 1, -2)$. These solutions correspond to the three assertions computed in Example 1:

$$k \geq 0 \ , \quad j \geq 0 \ , \quad \text{and} \quad j \geq 2k + 2 \ .$$

Any one of these solutions could be returned by the constraint solver.

On later iterations, χ_i consists of a conjunction of affine assertions. These assertions are added as additional rows in the tables, resulting in more λ-variables and more complicated dual constraints.

Farkas's Lemma states that the relationship between the primal and dual constraint systems is strict: the universal constraints of the primal system are valid if and only if the dual (existential) constraint system has a solution. Generalizing to polynomials preserves soundness but drops completeness.

Corollary 1 (Polynomial Lemma). *Consider the universal constraint system S of polynomial inequalities over real variables $\overline{x} = \{x_1, \ldots, x_m\}$:*

$$\begin{array}{rl} A: & \left\{ \begin{array}{l} a_{1,0} + \sum_{i=1}^{m} a_{1,i} t_i \geq 0 \\ \vdots \\ a_{n,0} + \sum_{i=1}^{m} a_{n,i} t_i \geq 0 \end{array} \right. \\ \hline C: & \quad c_0 + \sum_{i=1}^{m} c_i t_i \quad \geq 0 \end{array}$$

where the t_i are monomials over $\overline{x}$. That is, $S\colon\ \forall \overline{x}.\ A \ \rightarrow\ C$. Construct the dual constraint system as follows. For monomials t_i of even power (e.g., 1, x^2, x^2y^4, etc.), impose the constraint

$$c_i \geq \lambda_j a_{1,i} + \cdots + \lambda_n a_{n,i} \ ;$$

for all other terms, impose the constraint

$$c_i = \mu_j a_{1,i} + \cdots + \lambda_n a_{n,i} \ .$$

If the dual constraint system is satisfiable (for all $\lambda_j \geq 0$), then the primal constraint system is valid.

In particular, if the constraint system S is parameterized, then a solution to the dual constraint system provides an instantiation of the parameters that validates the primal constraints.

4.2 Solving Iterative VCPs

Farkas's Lemma and its polynomial generalization provide a method for solving the universal constraints of a VCP. However, the $\mathsf{strengthening}$ and Π-$\mathsf{strengthening}$ conditions impose existential constraints. One possibility is to ignore these existential constraints and apply the methods of [7] and [10]. The authors of [7] solve the constraints using a specialized solver described in [18]. The method of [10] is essentially restricted to verification constraints in which a single solution solves the problem (*e.g.*, to prove termination by finding a ranking function; see Section 5).

Unfortunately, dropping the existential constraints prevents us from performing iterative strengthening easily. Even with a strong χ_i, the constraint solver can always return the same solution that it has returned previously. For example, in Example 4, ignoring the $\mathsf{strengthening}$ constraint would allow the solution $k \geq 0$ to be returned again and again, even after setting $\chi_1 \stackrel{\text{def}}{=} k \geq 0$.

Instead, we describe a *sampling-based iterative method* in this section. Unlike the specialized solver of [18], this method is applicable to many types of constraint systems and allows applying off-the-shelf constraint solvers.

As in the examples of Section 3, suppose we have a parameterized assertion $I[\overline{p}, \overline{x}] \geq 0$, where $\overline{p}$ are the parameters and $\overline{x}$ are the system variables. On iteration i, we have already computed inductive assertion χ_{i-1} and would like to compute a stronger inductive assertion χ_i. Let $\psi_i[\overline{p}]$ be the universal constraints (arising from $\mathsf{initiation}$ and $\mathsf{consecution}$) of the ith VCP; that is, ψ_i does not include a $\mathsf{strengthening}$ or Π-$\mathsf{strengthening}$ constraint. Perform the following steps:

1. Solve the existential constraint system χ_{i-1} if the $\mathsf{strengthening}$ condition is imposed, or the constraint system $\chi_{i-1} \ \wedge \ \neg\Pi$ if the Π-$\mathsf{strengthening}$ condition is imposed. In the latter case, if the system does not have a solution, then declare that $\Box\Pi$ is invariant. Otherwise, the solution is a state s.
2. Solve the existential constraint system

$$I[\overline{p}, s] < 0 \ \wedge \ \mathsf{dual}(\psi_i[\overline{p}])$$

   for $\overline{p}$, where dual constructs the dual constraint system (recall that ψ_i contains only universal constraints). If a solution is not found, return to Step 1; if a Π-$\mathsf{strengthening}$ condition is imposed, possibly weaken it to a $\mathsf{strengthening}$ condition. Otherwise, the solution $\overline{q}$ is an assignment to $\overline{p}$.
3. Optimize the discovered solution $\overline{q}$ of $\overline{p}$. Let $J[\overline{p}, \overline{x}]$ be the non-constant part of I (*e.g.*, $p_1 j + p_2 k$ in $p_0 + p_1 j + p_2 k$). Then solve the following optimization problem for the parameter p_0 of I:

$$\begin{array}{l} \textbf{minimize } p_0 \\ \textbf{subject to} \\ \quad p_0 + J[\overline{q}, \overline{x}] \geq 0 \ \wedge \ \mathsf{dual}(\psi_i[p_0, q_1, \ldots, q_k]) \end{array}$$

   At most the value of q_0 from Step 2 is returned as the minimum of p_0. Set q_0 to the new solution.

4. Let $\chi_i \stackrel{\text{def}}{=} \chi_{i-1} \ \wedge \ I[\overline{q}, \overline{x}] \geq 0$.

Thus, Step 1 addresses the $\mathsf{strengthening}$ or Π-$\mathsf{strengthening}$ constraint, while Steps 2 and 3 address the other (universal) constraints.

The constraint problem of Step 1 can be solved in numerous ways, including using decision procedures or numerical constraint solvers for solving linear and semi-algebraic constraint problems. It is best if these solvers can be randomized so that a wide selection of sample points is possible. The constraint problem of Step 2 can be solved using linear or convex programming. If the Polynomial Lemma is used, then linear programming is sufficient. Many implementations of such constraint solvers are available.

Note that the strict inequality of Step 2 is easily handled by transforming the feasibility problem into an optimization problem:

$$\begin{array}{l} \textbf{maximize } \epsilon \\ \textbf{subject to} \\ \qquad I[\overline{p}, s] \leq -\epsilon \ \wedge \ \mathsf{dual}(\psi_i[\overline{p}]) \end{array}$$

The original system is feasible if and only if the maximum is positive. The optimization of Step 3 is inspired by the optimization that is performed in [21].

The inductive invariants constructed incrementally in Examples 1, 2, and 3 were obtained using this approach.

Example 5 (Simple). Consider the first iteration of solving the VCP of Examples 1 and 4. According to the steps of the procedure, we have the following:

1. Solve the constraint problem χ_0 (recall $\chi_0 \stackrel{\text{def}}{=} \mathsf{true}$), producing, for example, state $(j : -1, \ k : -3)$.
2. Solve

$$\underbrace{p_0 - p_1 - 3p_2 < 0}_{I[-1,-3] \ < \ 0} \ \wedge \ \underbrace{p_0 + 2p_1 \geq 0 \ \wedge \ p_1 \geq 0 \ \wedge \ 2p_1 + p_2 \geq 0}_{\text{dual system, simplified from Example 4}} .$$

   One solution is $(p_0 : 1, \ p_1 : 0, \ p_2 : 1)$, corresponding to $1 + k \geq 0$.
3. Optimize

$$\begin{array}{l} \textbf{minimize } p_0 \\ \textbf{subject to} \\ \qquad p_0 + k \geq 0 \ \wedge \ p_0 \geq 0 \end{array}$$

   The optimal value of p_0 is 0, corresponding to assertion $k \geq 0$.
4. Let $\chi_1 \stackrel{\text{def}}{=} \chi_0 \ \wedge \ k \geq 0$.

The assertion $\chi_1 : \ k \geq 0$ excludes the sample state $(j : -1, \ k : -3)$, as well as any other state in which $k < 0$. Thus, no future iteration can again discover $k \geq 0$ (or any weaker assertion).

On the next iteration, Step 1 could find, for example, sample point $(j : -1, \ k : 5)$, satisfying $\chi_1 : \ k \geq 0$. Then discovering inductive assertion $j \geq 0$ in Steps 2 and 3 eliminates this point.

5 Related Work

Set-Constraint Based Analysis Set constraint-based program analyses (see, *e.g.*, [2, 1]) pose classical program analysis problems — *e.g.*, standard dataflow equations, simple type inference, and monomorphic closure analysis — as set constraint problems and then solve them. Domains are discrete.

Ranking Function Synthesis Synthesis of affine expressions for verification purposes was first studied extensively in the context of ranking function synthesis. A function $\delta\colon \overline{x} \to \mathbb{Z}$ is a *ranking function* if

- if $\tau \in \mathcal{T}$ is enabled, then δ is nonnegative: $\varphi \;\wedge\; \rho_\tau \;\Rightarrow\; \delta \geq 0$; (bounded)
- δ decreases when any $\tau \in \mathcal{T}$ is taken: $\varphi \;\wedge\; \rho_\tau \;\Rightarrow\; \delta > \delta'$. (ranking)

Unlike in the consecution condition of inductive assertions, the parameterized expression appears only in the consequent of the bounded and ranking conditions. Examining Farkas's Lemma shows that synthesis of linear ranking functions over linear loops with real variables is therefore polynomial-time computable: the dual constraint system is linear and thus polynomial-time solvable.

[12] shows how to generate constraint systems over loops with linear assertional guards and linear assignments for which solutions are linear ranking functions. In [8, 9], it is observed that duality of linear constraints achieves efficient synthesis. [15] proves that this duality-based method is complete for single-path loops. [3] presents a complete method for the general case and shows how lexicographic linear ranking functions can also be computed efficiently in practice. In [10], the approach is generalized by using semidefinite programming to approximate the polynomial case.

Several extensions of the constraint-based approach have been explored. [4] extends the method to generate *polyranking* functions [6], which generalize ranking functions. A polyranking function need not always decrease. Finally, [5] addresses loops with integer variables.

Invariant Generation Invariant generation is harder than pure ranking function synthesis. [7] proposes using Farkas's Lemma and nonlinear constraint solving to generate affine invariants. A specialized solver for this method is described in [18]. [17] specializes the technique to Petri nets, for which the problem is efficiently solvable. In [20], polynomial equation invariants are generated using tools from algebra. [5] addresses loops with integer variables.

Analysis of Continuous and Hybrid Systems Lyapunov functions of continuous systems have been generated using convex optimization for over a decade (see, *e.g.*, [14]). In [19], the methods of [20] are adapted to generating polynomial equation invariants of hybrid systems. [16] introduces *barrier certificates*, which is a continuous analogue of program invariants. They describe a semidefinite relaxation that approximates consecution. [22] introduces a general method for constructing a *time elapse operator* for overapproximating the reachable space during a continuous mode of a hybrid system.

6 Conclusion

It is natural to view verification conditions as constraints on parameterized expressions. In the constraint context, we show that the two new conditions strengthening and Π-strengthening guide the incremental construction of inductive assertions. Unlike previous approaches to incremental strengthening, our proposed sampling-based incremental method applies standard constraint solvers. Thus, it directly benefits from progress in constraint solving. Additionally, the Π-strengthening condition facilitates property-directed invariant generation without imposing a fixed limit on the size of the discovered inductive assertion.

We have a simple implementation of the iterative technique. To avoid numerical issues involved in floating-point computations, we always use the Polynomial Lemma to produce a parametric-linear constraint problem in which only terms with parameters are nonlinear (they are bilinear: the product of a parameter and a dual λ-multiplier). We then solve this constraint problem by lazily instantiating λ-multipliers over a fixed set of values (typically, $\{0, 1\}$) and passing the linear part of the problem to a rational linear program solver. We thus obtain rational instantiations of the parameters. As solvers for convex programs improve, we can use stronger versions [10] of the Polynomial Lemma to obtain more invariants and to analyze nonlinear transition systems. The iterative method extends to applications of these solvers without modification.

References

1. Aiken, A. Introduction to set constraint-based program analysis. *Science of Computer Programming 35* (1999), 79–111.
2. Aiken, A., and Wimmers, E. Solving systems of set constraints. In *LICS* (1992), pp. 329–340.
3. Bradley, A. R., Manna, Z., and Sipma, H. B. Linear ranking with reachability. In *Proc.* 17^{th} *Intl. Conference on Computer Aided Verification (CAV)* (July 2005), K. Etessami and S. K. Rajamani, Eds., vol. 3576 of *LNCS*, Springer Verlag, pp. 491–504.
4. Bradley, A. R., Manna, Z., and Sipma, H. B. The polyranking principle. In *ICALP* (2005).
5. Bradley, A. R., Manna, Z., and Sipma, H. B. Termination analysis of integer linear loops. In *CONCUR* (2005).
6. Bradley, A. R., Manna, Z., and Sipma, H. B. Termination of polynomial programs. In *Proc. of Verification, Model Checking and Abstract Interpretation (VMCAI)* (Paris, France, January 2005), R. Cousot, Ed., vol. 3385 of *LNCS*, Springer Verlag.
7. Colón, M., Sankaranarayanan, S., and Sipma, H. Linear invariant generation using non-linear constraint solving. In *Computer Aided Verification* (July 2003), vol. 2725 of *LNCS*, Springer-Verlag, pp. 420–433.
8. Colón, M., and Sipma, H. Synthesis of linear ranking functions. In *7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (April 2001), T. Margaria and W. Yi, Eds., vol. 2031 of *LNCS*, Springer Verlag, pp. 67–81.

9. Colón, M., and Sipma, H. Practical methods for proving program termination. In *Proc. 14^{th} Intl. Conference on Computer Aided Verification* (2002), vol. 2404 of *LNCS*, Springer Verlag, pp. 442–454.
10. Cousot, P. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In *Proc. Verification, Model Checking, and Abstract Interpretation: 5^{th} International Conference (VMCAI)* (2005), pp. 1–24.
11. Cousot, P., and Halbwachs, N. Automatic discovery of linear restraints among the variables of a program. In 5^{th} *ACM Symp. Princ. of Prog. Lang.* (Jan. 1978), pp. 84–97.
12. Katz, S. M., and Manna, Z. A closer look at termination. *Acta Informatica 5*, 4 (1975), 333–352.
13. Manna, Z., and Pnueli, A. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.
14. Papachristodoulou, A., and Prajna, S. On the construction of lyapunov functions using the sum of squares decomposition. In *CDC* (2002).
15. Podelski, A., and Rybalchenko, A. A complete method for the synthesis of linear ranking functions. In *VMCAI* (2004), pp. 239–251.
16. Prajna, S., and Jadbabaie, A. Safety verification of hybrid systems using barrier certificates. In *HSCC* (2004), vol. 2993 of *LNCS*, Springer.
17. Sankaranarayanan, S., Sipma, H. B., and Manna, Z. Petri net analysis using invariant generation. In *Verification: Theory and Practice* (Taurmina, Italy, 2003), N. Derschowitz, Ed., vol. 2772 of *LNCS*, Springer Verlag, pp. 682–701.
18. Sankaranarayanan, S., Sipma, H. B., and Manna, Z. Constraint-based linear relations analysis. In 11^{th} *Static Analysis Symposium (SAS'2004)* (2004), vol. 3148 of *LNCS*, Springer-Verlag, pp. 53–68.
19. Sankaranarayanan, S., Sipma, H. B., and Manna, Z. Constructing invariants for hybrid systems. In *Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings* (2004), vol. 2993 of *LNCS*, Springer-Verlag, pp. 539–554.
20. Sankaranarayanan, S., Sipma, H. B., and Manna, Z. Non-linear loop invariant generation using Gröbner bases,. In 31^{th} *ACM Symp. Princ. of Prog. Lang.* (Venice, Italy, January 2004), pp. 318–329.
21. Sankaranarayanan, S., Sipma, H. B., and Manna, Z. Scalable analysis of linear systems using mathematical programming. In *Proc. of Verification, Model Checking and Abstract Interpretation (VMCAI)* (Paris, France, January 2005), R. Cousot, Ed., vol. 3385 of *LNCS*, Springer Verlag.
22. Sankaranarayanan, S., Sipma, H. B., and Manna, Z. Fixed point iteration for computing the time elapse operator. In *HSCC* (2006).
23. Schrijver, A. *Theory of Linear and Integer Programming.* Wiley, 1986.