Generalized Homogeneous Polynomials for Efficient Template-Based Nonlinear Invariant Synthesis

Kensuke Kojima[1,2], Minoru Kinoshita[1], and Kohei Suenaga[1,3]

[1] Kyoto University
[2] JST CREST
[3] JST PRESTO

Abstract. The *template-based* method is one of the most successful approaches to algebraic invariant synthesis. In this method, an algorithm designates a *template polynomial* p over program variables, generates constraints for $p = 0$ to be indeed an invariant, and solves the generated constraints. However, the template-based method often suffers from increasing template size if the degree of a template polynomial is set too high.

We propose a technique to improve the efficiency of template-based methods applied to higher-degree polynomials. Our method is based on the following finding: If an algebraic invariant exists, then there is a specific algebraic invariant that we call a *generalized homogeneous algebraic (GHA) invariant*, which is often smaller. This finding justifies to use only a smaller template that corresponds to a GHA invariant in invariant synthesis.

Concretely, we state our finding above formally based on the abstract semantics of an imperative program proposed by Cachera et al. Then, we modify their template-based invariant synthesis so that it manages only GHA invariants; this modification is proved to be sound. We also empirically demonstrate that the restriction to GHA invariants is useful; our implementation is comparable to theirs in their benchmark; it outperforms their implementation for programs that require a higher-degree template.

1 Introduction

We consider the following problem: Given a program c, discover a fact that holds at the end of c regardless of the initial state. This problem is called a *postcondition problem*. This paper considers the case where we are to discover a postcondition written as *algebraic condition* $p_1 = 0 \wedge \cdots \wedge p_n = 0$ where $p_1, \ldots, p_n$ are polynomials over program variables; this problem is a basis for static verification of functional correctness.

One approach to this problem is *invariant synthesis*, in which we are to compute a family of predicates P_l indexed by the program locations l such that

```
1: x := x_0; v := v_0; t := t_0;
2: while t − a ≠ 0 do
3:     (x, v, t) := (x + v dt, v − g dt − ρv dt, t + dt);
4: end while
5:
```

Fig. 1. Program c_{fall}, which models a falling mass point. The symbols in the program represent the following quantities: x is the position of the point, v is its speed, t is time, x_0 is the initial position, v_0 is the initial speed, t_0 is the initial value of the clock t, g is the acceleration rate $-g$, ρ is the friction coefficient, and dt is the discretization interval. The simultaneous substitution in the loop body numerically updates the values of x, v, and t. The values of x and t are computed using the differential equations $\frac{d}{dt}x = v$ and $\frac{d}{dt}t = 1$. Because the force applied by the air to the mass point is $-\rho v$, the differential equation for v is $\frac{d}{dt}v = -g - \rho v$.

P_l holds whenever the execution of c reaches l. The invariant associated with the end of c is a solution to the postcondition problem.

Because of its importance in static program verification, algebraic invariant synthesis has been intensively studied [**?**, 13, 15, 16]. Among the proposed techniques, a successful approach is the constraint-based method, in which invariant synthesis is reduced to a constraint-solving problem. During constraint generation, this method uses *templates*, polynomials over the program variables with unknown parameters at the coefficient positions, that typically represent the invariants [16]. The algorithm generates constraints that ensure the templates to be the invariants and obtains the invariants by solving the constraints[4].

Example 1. The program in Figure 1 models the behavior of a mass point with weight 1 and a constant acceleration rate; the program takes friction between the mass point and the air into account[5]. For this program, a postcondition that holds regardless of the initial state is $-gt + gt_0 - v + v_0 - x\rho + x_0\rho = 0$.

We describe how a template method would compute the postcondition in Example 1. The method described here differs from one we explore in this paper. This explanation suggests the flavor of a template method.

A template method generates a *template polynomial* over the program variables that represents an invariant at Line 4. Suppose the generated polynomial $p(x_0, v_0, t_0, x, v, t, a, dt, g, \rho)$ is of degree 2 over the variables: $p(x_0, v_0, t_0, x, v, t, a, dt, g, \rho) := a_1 + a_{t_0}t_0 + a_{x_0}x_0 + \cdots + a_{g\rho}g\rho$, where a_w is the coefficient parameter associated to the power product w. The procedure then generates constraints such that $p(x_0, v_0, t_0, x, v, t, a, dt, g, \rho) = 0$ is indeed an invariant at Line 4. The method proposed by Sankaranarayanan et al. [16] based on the Gröbner basis [4] generates the constraints as an equations on the parameters; a solution to the con-

[4] The constraint-based method by Cachera et al. [3], which is the basis of the current paper, uses a template also for other purpose. See Section 6 for detail.

[5] Although the guard condition $t - a \neq 0$ should be $t - a < 0$ in a real-world numerical program, we use this example for the presentation purposes.

straints gives $-gt + ga - v + v_0 - x\rho + x_0\rho = 0$ in this case, which is indeed an invariant at the end of c_{fall}.

One of the major bottlenecks of the template method is blowup of the size of a template. Blindly generating a template of degree-d for a degree parameter d limits the scalability of a template method for higher-degree invariants. For example, the program in Example 1 has an invariant $-gt^2 + gt_0^2 - 2tv + 2t_0v_0 + 2x - 2x_0 = 0$ at Line 4. This invariant requires synthesis of a degree-3 template, which has $\binom{10+3}{3} = 286$ monomials in this case.

We propose a hack to alleviate this bottleneck in the template methods. Our method is inspired by a rule of thumb in physics called the *principle of quantity dimension*: A physical law should not add two quantities with different *quantity dimensions* [1]. If we accept this principle, then, at least for a physically meaningful program such as c_{fall}, an invariant (and therefore a template) should consist of the monomials with the same quantity dimensions.

Indeed, the polynomial $-gt + gt_0 - v + v_0 - x\rho + x_0\rho$ in the invariant we calculated in Example 1 consists only of the quantities that represent velocities. (Notice that ρ is a quantity that corresponds to the inverse of a time quantity.) The polynomial $-gt^2 + gt_0^2 - 2tv + 2t_0v_0 + 2x - 2x_0$ above consists only of quantities corresponding to distances. If we use the notation of quantity dimensions used in physics, the former polynomial consists only of monomials with the quantity dimension LT^{-1}, whereas the latter consists only of L, where L and T represent quantity dimensions for lengths and times, respectively. By leveraging the quantity dimension principle in the template synthesis phase, we can reduce the size of a template. For example, we could use a template that consists only of the monomials for, say, velocity quantities instead of $p(x_0, v_0, x, v, t, a, dt, g, \rho)$, which yields a smaller template.

The idea of the quantity dimension principle can be nicely captured by generalizing the notion of homogeneous polynomials. A polynomial is said to be homogeneous if it consists of monomials of the same degree; for example, the polynomial $x^3 + x^2y + xy^2 + y^3$ is a homogeneous polynomial of degree 3. We generalize this notion of homogeneity so that (1) a *degree* is an expression corresponding to a quantity dimension (e.g., LT^{-1}) and (2) each variable has its own degree in degree computation.

We describe our idea using an example, deferring the formal definitions. Suppose we have the following *degree assignment* to each program variable: $\Gamma := \{\, g \mapsto LT^{-2}, t \mapsto T, x \mapsto L, v \mapsto LT^{-1}, v_0 \mapsto LT^{-1}, \rho \mapsto T^{-1}, a \mapsto T \,\}$. This degree assignment intuitively corresponds to assignment of the quantity dimension to each variable. With this degree assignment Γ, all the monomials in $-gt + gt_0 - v + v_0 - x\rho + x_0\rho$ have the same degree; for example, the monomial $-gt$ has the degree $\Gamma(g)\Gamma(t) = (LT^{-2})T = LT^{-1}$ and the monomial $x\rho$ has $\Gamma(x)\Gamma(\rho) = LT^{-1}$, and so on. Hence, $-gt + gt_0 - v + v_0 - x\rho + x_0\rho$ is a homogeneous polynomial in the generalized sense. Such a polynomial is called a *generalized homogeneous (GH) polynomial*. We call an algebraic invariant with a GH polynomial a *generalized homogeneous algebraic (GHA) invariant*.

The main result of this paper is the following: If there is an algebraic invariant of a given program c, then there is a GHA invariant. This justifies the use use of a template that corresponds to a GH polynomial in template method. We show this result by using the abstract semantics of an imperative programming language proposed by Cachera et al. [3]. We empirically show that the algorithm by Cachera et al. can indeed be made efficient using the idea of GH polynomials.

As we saw above, the definition of GH polynomials is parameterized over a degree assignment Γ. We found that the type inference algorithm for the *dimension type system* proposed by Kennedy [11,12] can be used to find an appropriate degree assignment; for example, Γ above is inferred using this algorithm. The dimension type system was originally proposed for detecting a violation of the quantity-dimension principle in a numerical program. We show that this type system is also useful for invariant synthesis.

Although the method is inspired by the principle of quantity dimensions, our method can be applied to a program that does not model a physical phenomenon because we abstract the notion of a quantity dimension using that of generalized homogeneity; all the programs used in our experiments (Section 7) are indeed physically nonsensical programs.

The rest of this paper is organized as follows. Section 2 sets up the basic mathematical definitions used in this paper; Section 3 defines the syntax and semantics of the target language and its abstract semantics; Section 4 defines GH polynomials; Section 5 defines the revised abstract semantics as the restriction of the original one to the set of GH polynomials and shows that the revised semantics is sound and complete; Section 6 gives a template-based invariant-synthesis algorithm and shows its soundness; Section 7 reports the experimental results; Section 8 discusses related work; and Section 9 presents the conclusions. Several proofs are given in the appendices.

2 Preliminary

$\mathbb{R}$ is the set of real numbers and $\mathbb{N}$ is the set of natural numbers. We write $|S|$ for the cardinality of S if S is a finite set. We designate an infinite set of *variables* $\mathbf{Var}$. K is a field ranged over by metavariable k; we use the standard notation for the operations on K. For $x_1, \ldots, x_n \in \mathbf{Var}$, we write $K[x_1, \ldots, x_n]$, ranged over by p and q, for the set of polynomials over $x_1, \ldots, x_n$, the coefficients of which are taken from K; it is a ring with the standard addition and multiplication.

A subset $I \subseteq K[x_1, \ldots, x_n]$ is called an *ideal* if (1) I is an additive subgroup and (2) $pq \in I$ for any $p \in I$ and $q \in K[x_1, \ldots, x_n]$. A set $S \subseteq K[x_1, \ldots, x_n]$ is said to *generate* the ideal I (equivalently, S is a *generator* of I) if I is the smallest ideal that contains S. The following facts are paramount: (1) there is a unique ideal generated by S for a set $S \subseteq K[x_1, \ldots, x_n]$, and (2) every ideal I has a finite generator [4]. We write $\langle S \rangle$ for the ideal generated by S.

We call an expression of the form $x_1^{d_1} \ldots x_N^{d_N}$, where $d_1, \ldots, d_N \in \mathbb{N}$ and $x_1, \ldots, x_N \in \mathbf{Var}$, a *power product* over $x_1, \ldots, x_n$; w is a metavariable for power products. we call $\sum d_i$ the *degree* of this power product. A *monomial*

is a term of the form kw; the degree of this monomial is that of w. We write $\mathbf{deg}(p)$, that is, the degree of the polynomial p, for the maximum degree of the monomials in p.

A *state*, ranged over by σ, is a finite map from $\mathbf{Var}$ to K. We write $\mathbf{St}$ for the set of states. We use the metavariable S for a subset of $\mathbf{St}$. We write $\sigma(p)$, where $p \in K[x_1, \ldots, x_n]$, for $p(\sigma(x_1), \ldots, \sigma(x_n))$. The set $\mathcal{P}(\mathbf{St})$ constitutes a complete lattice with respect to the set-inclusion order.

3 Language

This section defines the target language, its concrete semantics, and its abstract semantics. We essentially follow the development by Cachera et al. [3]; we refer the interested reader to this paper.

The syntax of the target language is as follows:

$$c ::= \mathbf{skip} \mid x{:=}p \mid c_1; c_2 \mid \mathbf{if}\, p = 0\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2 \mid \mathbf{while}\ p = 0\ \mathbf{do}\ c \mid \mathbf{while}\ p \neq 0\ \mathbf{do}\ c$$

where p is a polynomial over the program variables. We restrict the guard to an algebraic condition (i.e., $p = 0$), or its negation.

The semantics of this language is given by the following denotation function, which is essentially the same as that of Cachera et al.

$$\begin{aligned}
[\![c]\!] &: (\mathcal{P}(\mathbf{St}), \subseteq) \to (\mathcal{P}(\mathbf{St}), \subseteq) \\
[\![\mathbf{skip}]\!](S) &= S \\
[\![x{:=}p]\!](S) &= \{\, \sigma \mid \sigma[x \mapsto \sigma(p)] \in S \,\} \\
[\![c_1; c_2]\!](S) &= [\![c_1]\!]([\![c_2]\!](S)) \\
[\![\mathbf{if}\, p = 0\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2]\!](S) &= \{\, \sigma \in [\![c_1]\!](S) \mid \sigma(p) = 0 \,\} \cup \{\, \sigma \in [\![c_2]\!](S) \mid \sigma(p) \neq 0 \,\} \\
[\![\mathbf{while}\ p \bowtie 0\ \mathbf{do}\ c]\!](S) &= \nu(\lambda X.\, \{\, \sigma \in S \mid \sigma(p) \not\bowtie 0 \,\} \cup \{\, \sigma \in [\![c]\!](X) \mid \sigma(p) \bowtie 0 \,\}),
\end{aligned}$$

where $\bowtie \in \{ =, \neq \}$ and νF is the greatest fixed point of F. Intuitively, $\sigma \in [\![c]\!](S)$ means that executing c from σ results in a state in S if the execution terminates; notice that σ should be in $[\![c]\!](S)$ if c does not terminate. We use the greatest fixed point instead of the least fixed point in the **while** statement so that $[\![c]\!](S)$ contains the states from which the execution of c does not terminate; if we used the least fixed point in the semantics of a while loop, then only the initial states from which the program terminates would be in the denotation of the loop. For example, consider the following program P that does not terminate for any initial state: **while** $0 = 0$ **do skip**. Then, $[\![P]\!](S)$ should be $\mathbf{St}$. However, if the denotation of a **while** loop were given by the least fixed point, then $[\![P]\!](S)$ would be $\emptyset$.

Example 2. We write c_1 for $(x, v, t) := (x_0, v_0, t_0)$, c_2 for $(x, v, t) := (x + v\,dt, v - g\,dt - \rho v\,dt, t + dt)$, p_1 for $-gt + gt_0 - v + v_0 - x\rho + x_0\rho$, p_2 for $-gt^2 + gt_0^2 - 2tv + 2t_0v_0 + 2x - 2x_0$, and p for $p_1 + p_2$. Let $S = \{\, \sigma \in \mathbf{St} \mid \sigma(p) = 0 \,\}$. We show that $[\![c_{fall}]\!](S) = \mathbf{St}$. We have $[\![c_{fall}]\!](S) = [\![c_1]\!]([\![\mathbf{while}\ t - a \neq 0\ \mathbf{do}\ c_2]\!](S)) = [\![c_1]\!](\nu F)$ where $F(X) = \{\, \sigma \in S \mid \sigma(t - a) = 0 \,\} \cup \{\, \sigma \in [\![c_2]\!](X) \mid \sigma(t - a) \neq 0 \,\}$. It is easy to check that $[\![c_1]\!](S) = \mathbf{St}$, so it suffices to show that $\nu F \supseteq S$. Note that

$[\![c_2]\!](S) = S$ because c_2 does not change the value of p. From this we obtain the following as desired: $F(S) = \{\, \sigma \in S \mid \sigma(t-a) = 0 \,\} \cup \{\, \sigma \in [\![c_2]\!](S) \mid \sigma(t-a) \neq 0 \,\} = \{\, \sigma \in S \mid \sigma(t-a) = 0 \,\} \cup \{\, \sigma \in S \mid \sigma(t-a) \neq 0 \,\} = S$.

We give an abstract semantics of this language. It is essentially the same as that given by Cachera et al. [3] with a small adjustment in the presentation.

We first define the abstract domain that we use. The preorder $\sqsubseteq^\sharp \subseteq \mathcal{P}(K[x_1, \ldots, x_n]) \times \mathcal{P}(K[x_1, \ldots, x_n])$ is defined by $S_1 \sqsubseteq^\sharp S_2 :\iff S_2 \subseteq S_1$[6]. Then $\mathcal{P}(K[x_1, \ldots, x_n])$ is a complete lattice, and meets are given by unions of sets: Given $H \in \mathcal{P}(K[x_1, \ldots, x_n])$ and $U \subseteq \mathcal{P}(K[x_1, \ldots, x_n])$, $H \sqsubseteq^\sharp G$ for all $G \in U$ if and only if $H \sqsubseteq^\sharp \bigcup U$.

We define the abstraction $\alpha(S)$ by $\{\, p \in K[x_1, \ldots, x_n] \mid \forall \sigma \in S, \sigma(p) = 0 \,\}$ and the concretization $\gamma(G)$ by $\{\, \sigma \in \mathbf{St} \mid \forall p \in G, \sigma(p) = 0 \,\}$. The pair of α and γ constitutes a Galois connection; indeed, $\alpha(S) \sqsubseteq^\sharp G$ if and only if $S \subseteq \gamma(G)$, because by definition both of them are equivalent to: $\forall p \in G, \forall \sigma \in S, \sigma(p) = 0$. The discussion above also gives an intuition of our abstraction; a set of states S is abstracted by a set of polynomials $G := \{\, p_1, \ldots, p_m \,\}$ if S is the set of the solutions of the equation $p_1 = 0 \wedge \cdots \wedge p_m = 0$. For example, the set of a state $\{\, \{\, x_1 \mapsto 1, x_2 \mapsto 0 \,\} \,\}$ is abstracted by the set $\{\, (x_1 - 1)p_1 + x_2 p_2 \mid p_1, p_2 \in K[x_1, \ldots, x_n] \,\}$; this set is equivalently $\langle x_1 - 1, x_2 \rangle$.

The definition of the abstract semantics is parameterized over a *remainder* operation $\mathbf{Rem}(f, p)$ that satisfies $\mathbf{Rem}(f, p) = f - qp$ for some q. Note that this definition differs from that in the standard multivariate polynomial algebra, where we need to fix a monomial order $\preceq$ so that the operation is well-defined, and we need to force that $\mathrm{LM}(p)$ does not divide any monomial in $\mathrm{LM}(\mathbf{Rem}(f, p))$, where $\mathrm{LM}(r)$ is the monomial in r that is the greatest with respect to the order $\preceq$; we here do not require such conditions. We write $\mathbf{Rem}(G, p)$, where G is a set of polynomials, for the set $\{\, \mathbf{Rem}(f, p) \mid f \in G \setminus \{\, 0 \,\} \,\}$.

The abstract semantics $[\![c]\!]^\sharp_{\mathbf{Rem}}$ is defined as follows.

$$
\begin{aligned}
[\![c]\!]^\sharp_{\mathbf{Rem}} &: (\mathcal{P}(K[x_1, \ldots, x_n]), \sqsubseteq^\sharp) \to (\mathcal{P}(K[x_1, \ldots, x_n]), \sqsubseteq^\sharp) \\
[\![\mathbf{skip}]\!]^\sharp_{\mathbf{Rem}}(G) &= G \\
[\![x{:=}p]\!]^\sharp_{\mathbf{Rem}}(G) &= G[x := p] \\
[\![c_1; c_2]\!]^\sharp_{\mathbf{Rem}}(G) &= [\![c_1]\!]^\sharp_{\mathbf{Rem}}([\![c_2]\!]^\sharp_{\mathbf{Rem}}(G)) \\
[\![\mathbf{if}\, p = 0\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2]\!]^\sharp_{\mathbf{Rem}}(G) &= p \cdot [\![c_2]\!]^\sharp_{\mathbf{Rem}}(G) \cup \mathbf{Rem}([\![c_1]\!]^\sharp_{\mathbf{Rem}}(G), p) \\
[\![\mathbf{while}\, p \neq 0\, \mathbf{do}\, c]\!]^\sharp_{\mathbf{Rem}}(G) &= \nu(\lambda H. p \cdot [\![c_1]\!]^\sharp_{\mathbf{Rem}}(H) \cup \mathbf{Rem}(G, p)) \\
[\![\mathbf{while}\, p = 0\, \mathbf{do}\, c]\!]^\sharp_{\mathbf{Rem}}(G) &= \nu(\lambda H. p \cdot G \cup \mathbf{Rem}([\![c_1]\!]^\sharp_{\mathbf{Rem}}(H), p)).
\end{aligned}
$$

$G[x := p] = \{\, q[x := p] \mid q \in G \,\}$ and $q[x := p]$ is the polynomial obtained by replacing x with p in q. νF denotes the greatest fixed point of F, which exists for an arbitrary monotone F because we are working in the complete lattice $\mathcal{P}(K[x_1, \ldots, x_n])$; concretely, we have $\nu F = \bigcup \{\, G \mid G \sqsubseteq^\sharp F(G) \,\}$.

$[\![c]\!]^\sharp_{\mathbf{Rem}}$ transfers backward a set of polynomials with value 0; we call such polynomial a *zero polynomial*. The highlight of the abstract semantics is the

[6] The original abstract semantics of Cachera et al. [3] is defined as a transformer on *ideals* of polynomials; however, we formulate it here so that it operates on *sets* of polynomials because their invariant-synthesis algorithm depends on the choice of a generator of an ideal.

definition of $[\![\mathbf{if}\, p = 0\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2]\!]^\sharp_{\mathbf{Rem}}(G)$. To understand this case, first notice that $[\![c_1]\!]^\sharp_{\mathbf{Rem}}(G)$ is a set of zero polynomials just before c_1 is executed; $[\![c_2]\!]^\sharp_{\mathbf{Rem}}(G)$ is those for c_2. Because we know that $p = 0$ holds just before c_1, the set of zero polynomials should include $\mathbf{Rem}([\![c_1]\!]^\sharp_{\mathbf{Rem}}(G), p)$. The reader is referred to [3] for a more detailed explanation.

The abstract semantics is related to the postcondition problem as follows: If $[\![c]\!]^\sharp_{\mathbf{Rem}}(G) = \{\, 0 \,\}$, then $[\![c]\!](\gamma(G)) = \mathbf{St}$. Indeed, from the soundness above, $\gamma([\![c]\!]^\sharp_{\mathbf{Rem}}(G)) = \gamma(\{\, 0 \,\}) = \mathbf{St} \subseteq [\![c]\!](\gamma(G))$; therefore $[\![c]\!](\gamma(G)) = \mathbf{St}$ follows because $\mathbf{St}$ is the top element in the concrete domain.

Example 3. We exemplify how the abstract semantics works using the program c_{fall} in Figure 1. Set p, c_1, and c_2 as in Example 2. Define $\mathbf{Rem}$ in this example by $\mathbf{Rem}(f, p) = f$. The following calculation shows $[\![c_{fall}]\!]^\sharp_{\mathbf{Rem}}(\{\, p \,\}) = \{\, 0 \,\}$: $[\![c_{fall}]\!]^\sharp_{\mathbf{Rem}}(\{\, p \,\}) = [\![c_1]\!]^\sharp_{\mathbf{Rem}}([\![\mathbf{while}\ t{-}a \neq 0\ \mathbf{do}\ c_2]\!]^\sharp_{\mathbf{Rem}}(\{\, p \,\})) = [\![c_1]\!]^\sharp_{\mathbf{Rem}}(\nu(\lambda H.(t{-}a)[\![c_2]\!]^\sharp_{\mathbf{Rem}}(H) \cup \mathbf{Rem}(\{\, p \,\}, t{-}a))) = [\![c_1]\!]^\sharp_{\mathbf{Rem}}(\nu(\lambda H.(t{-}a)[\![c_2]\!]^\sharp_{\mathbf{Rem}}(H) \cup \{\, p \,\})) = [\![c_1]\!]^\sharp_{\mathbf{Rem}}(\{\, (t-a)^n p \mid n \in \mathbb{N} \,\}) = \{\, ((t-a)^n p)[x := x_0, v := v_0, t := t_0] \mid n \in \mathbb{N} \,\} = \{\, 0 \,\}$. The greatest fixed point is computed as follows. Let $F(H) = (t{-}a)[\![c_2]\!]^\sharp_{\mathbf{Rem}}(H) \cup \{\, p \,\}$ and $G = \{\, (t-a)^n p \mid n \in \mathbb{N} \,\}$. We show that $\nu F = G$. According to the definition of $\sqsubseteq^\sharp$, we have $\top = \emptyset$, and it is easy to check that $F^n(\top) = \{\, (t-a)^k p \mid 0 \leq k \leq n \,\}$. Therefore, noting νF is the limit of $(F^n(\top))_{n \in \mathbb{N}}$, we have $\nu F \sqsubseteq^\sharp G$. By simple computation, we can see that G is a fixed point of F, so we also have $G \sqsubseteq^\sharp \nu F$; hence, $\nu F = G$.

Cachera et al. [3, Theorem 3] showed the soundness of this abstract semantics: For any program c and a set of polynomials G, we have $\gamma([\![c]\!]^\sharp_{\mathbf{Rem}}(G)) \subseteq [\![c]\!](\gamma(G))$. Although our abstract value is a set, instead of an ideal of polynomials, we can prove this theorem in the same way as the original proof.

4 Generalized homogeneous polynomials

A polynomial p is said to be a homogeneous polynomial of degree d if the degree of every monomial in p is d [4]. As we mentioned in Section 1, we generalize this notion of homogeneity.

We first generalize the notion of the degree of a polynomial.

Definition 1. *The group of* generalized degrees (g-degrees) $\mathbf{GDeg}_B$*, ranged over by* τ*, is an Abelian group freely generated by the finite set* B*; that is,* $\mathbf{GDeg}_B := \{\, b_1^{n_1} \dots b_m^{n_m} \mid b_1, \dots, b_m \in B, n_1, \dots, n_m \in \mathbb{N} \,\}$*. We call* B *the set of the* base degrees*. We often omit* B *in* $\mathbf{GDeg}_B$ *if the set of the base degrees does not matter.*

For example, if we set B to $\{\, L, T \,\}$, then L, T, and LT^{-1} are all generalized degrees. By definition, $\mathbf{GDeg}_B$ has the multiplication on these g-degrees (e.g., $(LT) \cdot (LT^{-2}) = L^2T^{-1}$ and $(LT^2)^2 = L^2T^4$.)

$$\frac{}{\Gamma \vdash \mathbf{skip}} \quad \text{(T-Skip)}$$

$$\frac{\Gamma \vdash c_1 \qquad \Gamma \vdash c_2}{\Gamma \vdash c_1; c_2} \quad \text{(T-Seq)}$$

$$\frac{\Gamma(x) = \mathbf{gdeg}_\Gamma(p)}{\Gamma \vdash x{:=}p} \quad \text{(T-Assign)}$$

$$\frac{\mathbf{gdeg}_\Gamma(p) = \tau \qquad \Gamma \vdash c_1 \qquad \Gamma \vdash c_2}{\Gamma \vdash \mathbf{if}\, p = 0\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2} \quad \text{(T-If)}$$

$$\frac{\mathbf{gdeg}_\Gamma(p) = \tau \qquad \Gamma \vdash c}{\Gamma \vdash \mathbf{while}\ p \bowtie 0\ \mathbf{do}\ c} \quad \text{(T-While)}$$

Fig. 2. Typing rules

In the analogy of quantity dimensions, the set B corresponds to the base quantity dimensions (e.g., L for lengths and T for times); the set $\mathbf{GDeg}_B$ corresponds to the derived quantity dimensions (e.g., LT^{-1} for velocities and LT^{-2} for acceleration rates.); multiplication expresses the relationship among quantity dimensions (e.g., $LT^{-1} \cdot T = L$ for velocity × time = distance.)

Definition 2. *A* g-degree assignment *is a finite mapping from* **Var** *to* **GDeg**. *A metavariable Γ ranges over the set of g-degree assignments. For a power product $w := x_1^{d_1} \dots x_n^{d_n}$, we write $\mathbf{gdeg}_\Gamma(w)$ for $\Gamma(x_1)^{d_1} \dots \Gamma(x_n)^{d_n}$ and call it the* g-degree of w under Γ *(or simply* g-degree *of w if Γ is not important); $\mathbf{gdeg}_\Gamma(kw)$, the g-degree of a monomial kw under Γ, is defined by $\mathbf{gdeg}_\Gamma(w)$.*

For example, set Γ to $\{\, t \mapsto T, v \mapsto LT^{-1} \,\}$; then $\mathbf{gdeg}_\Gamma(2vt) = L$. In terms of the analogy with quantity dimensions, this means that the expression $2vt$ represents a length.

Definition 3. *We say p is a* generalized homogeneous (GH) polynomial of g-degree τ under Γ *if every monomial in p has the g-degree τ under Γ. We write $\mathbf{gdeg}_\Gamma(p)$ for the g-degree of p if it is a GH polynomial under Γ; if it is not, then $\mathbf{gdeg}_\Gamma(p)$ is not defined. We write $K[x_1, \dots, x_n]_{\Gamma,\tau}$ for the set of the GH polynomials with g-degree τ under Γ. We write $K[x_1, \dots, x_n]_\Gamma$ for $\bigcup_{\tau \in \mathbf{GDeg}} K[x_1, \dots, x_n]_{\Gamma,\tau}$.*

Example 4. The polynomial $-gt^2 + gt_0^2 - 2tv + 2t_0v_0 + 2x - 2x_0$ (the polynomial p_2 in Example 2) is a GH-polynomial under

$$\Gamma := \{\, g \mapsto LT^{-2}, t \mapsto T, v \mapsto LT^{-1}, x \mapsto L, x_0 \mapsto L, v_0 \mapsto LT^{-1}, \rho \mapsto T^{-1}, a \mapsto T \,\}$$

because all the monomials in p_2 have the same g-degree in common; for example, $\mathbf{gdeg}_\Gamma(-gt^2) = \Gamma(g)\Gamma(t)^2 = (LT^{-2})T^2 = L$; $\mathbf{gdeg}_\Gamma(-2tv) = \Gamma(t)\Gamma(v) = T(LT^{-1}) = L$; $\mathbf{gdeg}_\Gamma(2x) = \Gamma(x) = L$; and $\mathbf{gdeg}_\Gamma(-2x_0) = \Gamma(x_0) = L$. Therefore, $\mathbf{gdeg}_\Gamma(p_2) = L$. We also have $\mathbf{gdeg}_\Gamma(p_1) = LT^{-1}$.

It is easy to see that any $p \in K[x_1, \ldots, x_n]$ can be uniquely written as the finite sum of GH polynomials as $p_{\Gamma,\tau_1} + \cdots + p_{\Gamma,\tau_m}$, where p_{Γ,τ_i} is the summand of g-degree τ_i under Γ in this representation. For example, the polynomial p in Example 2, can be written as $p_L + p_{LT^{-1}}$ where $p_L = p_1$ and $p_{LT^{-1}} = p_2$ from the previous example. We call $p_{\Gamma,\tau}$ the *GH component of p with g-degree τ under Γ*; we often omit Γ part if it is clear from the context.

The definitions above are parameterized over a g-degree assignment Γ. It is determined from the usage of variables in a given program, which is captured by the following type judgment.

Definition 4. *The judgment $\Gamma \vdash c$ is the smallest relation that satisfies the rules in Figure 2. We say Γ is* consistent *with the program c if $\Gamma \vdash c$ holds.*

The consistency relation above is an adaptation of the *dimension type system* proposed by Kennedy [11,12] to our imperative language. A g-degree assignment Γ such that $\Gamma \vdash c$ holds makes every polynomial in c a GH one. In the rule T-Assign, we require the polynomial p to have the same g-degree as that of x in Γ.

Example 5. Set Γ as in Example 4. From the definition of $\mathbf{gdeg}$, we have $\mathbf{gdeg}_\Gamma(x + vdt) = L$, $\mathbf{gdeg}_\Gamma(v - gdt - \rho v dt) = LT^{-1}$, and $\mathbf{gdeg}_\Gamma(t + dt) = T$; therefore, we have $\Gamma \vdash (x, v, t) := (x + vdt, v - gdt - \rho v dt, t + dt)$ from T-Assign. From $\mathbf{gdeg}_\Gamma(t - a) = T$, we have $\Gamma \vdash \mathbf{while}\ t - a \neq 0\ \mathbf{do}\ (x, v, t) := (x + vdt, v - gdt - \rho v dt, t + dt)$;. From $\Gamma \vdash (x, v, t) := (x_0, v_0, t_0)$ and T-Seq, we have $\Gamma \vdash c_{fall}$.

5 Abstract semantics restricted to GH polynomials

This section gives the main result of this paper: If there is an algebraic invariant of c and $\Gamma \vdash c$, then there exists an algebraic invariant that consists of a GH polynomial under Γ.

To state this result formally, we revise our abstract semantics by restricting it to the domain of the GH polynomials. The domain is obtained by replacing the underlying set of the domain $(\mathcal{P}(K[x_1, \ldots, x_n]), \sqsubseteq^\sharp)$ with $\mathcal{P}(K[x_1, \ldots, x_n]_\Gamma)$. This is a subset of $\mathcal{P}(K[x_1, \ldots, x_n])$ that is closed under arbitrary meets. We can define the abstraction and the concretization in the same way as in Section 3.

The revised abstract semantics $[\![c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}$, which we hereafter call *GH abstract semantics*, is the same as the original one except that it is parameterized over the degree assignment Γ. In the following definition, we write $\mathbf{Rem}_\Gamma(G, p)$ for the set of the remainder obtained from a GH polynomial in G and p: $\mathbf{Rem}_\Gamma(G, p) := \{\, \mathbf{Rem}(f, p) \mid f \in (G \cap K[x_1, \ldots, x_n]_\Gamma) \setminus \{\, 0 \,\} \,\}$. We assume that our choice of $\mathbf{Rem}$ is a remainder operation such that whenever both f and p are GH poly-

nomials, so is $\mathbf{Rem}(f,p)$.

$$\begin{aligned}
[\![\mathbf{skip}]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(G) &= G\\
[\![x{:=}p]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(G) &= G[x := p]\\
[\![c_1; c_2]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(G) &= [\![c_1]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}([\![c_2]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(G))\\
[\![\mathbf{if}\, p = 0\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(G) &= p \cdot [\![c_2]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(G) \cup \mathbf{Rem}_\Gamma([\![c_1]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(G), p)\\
[\![\mathbf{while}\ p \neq 0\ \mathbf{do}\ c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(G) &= \nu(\lambda H.p \cdot [\![c_1]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(H) \cup \mathbf{Rem}_\Gamma(G,p))\\
[\![\mathbf{while}\ p = 0\ \mathbf{do}\ c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(G) &= \nu(\lambda H.p \cdot G \cup \mathbf{Rem}_\Gamma([\![c_1]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(H), p)).
\end{aligned}$$

The following proposition guarantees the existence of $\mathbf{Rem}_\Gamma$.

Proposition 1. *Let $f, p \in K[x_1, \ldots, x_n]_\Gamma$ and $f = pq + r$ for some $q, r \in K[x_1, \ldots, x_n]$ (n.b., q and r are not necessarily GH); then there exist homogeneous components q' of q and r' of r such that $f = pq' + r'$.*

Proof. Set q' to $q_{\mathbf{gdeg}_\Gamma(f)\mathbf{gdeg}_\Gamma(p)^{-1}}$ and r' to $r_{\mathbf{gdeg}_\Gamma(f)}$.

The following theorem guarantees that the invariant found using the semantics $[\![c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}$ is indeed an invariant of c.

Theorem 1 (Soundness of the GH abstract semantics).
If $\Gamma \vdash c$ and G is the set of GH polynomials under Γ, then $[\![c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(G) = [\![c]\!]^{\sharp}_{\mathbf{Rem}}(G)$.

Proof. By induction on c. Notice that $\mathbf{Rem}(G,p) = \mathbf{Rem}_\Gamma(G,p)$ if G and p are homogeneous under Γ under the assumption that **Rem** preserves homogeneity.

This theorem implies that if g is a GH polynomial under Γ and $[\![c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(g) = 0$, then g is indeed an invariant.

Completeness of $[\![c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}$ is obtained as a corollary of the following lemma.

Lemma 1. *Suppose $\Gamma \vdash c$ and $g'_1, \ldots, g'_m \in K[x_1, \ldots, x_n]$. Further, suppose that g_i is a homogeneous component of g'_i (i.e., $g_i = g'_{i\tau_i}$ for some τ_i). If $h \in [\![c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(\{\, g_1, \ldots, g_m \,\})$, then there exists $h' \in [\![c]\!]^{\sharp}_{\mathbf{Rem}}(\{\, g'_1, \ldots, g'_m \,\})$ such that h is a homogeneous component of h'.*

Proof. We say G is a homogeneous component of G' under Γ if, for any $p \in G$, there exists $p' \in G'$ such that $p = p'_\tau$ for some τ. By induction on c, we can prove that if G is a homogeneous component of G' under Γ, then $[\![c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(G)$ is a homogeneous component of $[\![c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(G')$ under Γ.

Theorem 2 (Completeness). *Let g_i and g'_i be the same as in Lemma 1. If $\Gamma \vdash c$ and $[\![c]\!]^{\sharp}_{\mathbf{Rem}}(\{\, g'_1, \ldots, g'_m \,\}) = \{\, 0 \,\}$, then $[\![c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(\{\, g_1, \ldots, g_m \,\}) = \{\, 0 \,\}$.*

Proof. Take $h \in [\![c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}(\{\, g_1, \ldots, g_m \,\})$. Then there exists $h' \in [\![c]\!]^{\sharp}_{\mathbf{Rem}}(\{\, g'_1, \ldots, g'_m \,\})$ such that $h'_{\mathbf{gdeg}(h)} = h$. By assumption we have $h' = 0$; therefore $h = 0$.

Hence, if g is an invariant of c in the sense of Cachera et al., then every homogeneous component of g is also an invariant.

Example 6. Example 3, Example 5, and Corollary 2 guarantee that the following equations: $[\![c]\!]^{\sharp}_{\mathbf{Rem}}\{\, p_1 \,\} = \{\, 0 \,\}$ and $[\![c]\!]^{\sharp}_{\mathbf{Rem}}\{\, p_2 \,\} = \{\, 0 \,\}$. One can directly confirm these equations in the same way as Example 3.

6 Template-based algorithm

This section applies our idea to Cachera's tempalte-based invariant-synthesis algorithm [3]. We hereafter use metavariable a for a *parameter* that represents an unknown value. We use metavariable A for a set of parameters. A *template* on A is an expression of the form $a_1p_1 + \cdots + a_np_n$; we use metavariable G for a set of templates. We denote the set of templates on A by $T(A)$. A *valuation* v on A is a map from A to K. We can regard v as a map from $T(A)$ to $K[x_1, \ldots, x_n]$ by $v(a_1p_1 + \cdots + a_mp_m) = v(a_1)p_1 + \cdots + v(a_m)p_m$.

6.1 Algorithm proposed by Cachera et al.

Cachera et al. proposed a sound template-based algorithm for the postcondition problem we mentioned in Section 1 by using the abstract semantics $[\![c]\!]^\sharp_{\mathbf{Rem}}$ and without using the Gröbner basis. Their basic idea is to express a fixed point by constraints on the parameters in a template in order to avoid fixed-point iteration.

To recall the algorithm of Cachera et al., we establish several definitions.

Definition 5. *An* equality constraint *on A is a pair of G and G', denoted as $\langle G \equiv G' \rangle$, where $G, G' \subseteq T(A)$. A* constraint set *on A, or just* constraints*, is a set of equality constraints on A; a constraint set is represented by the metavariable C. We may write (A, C) for a constraint set C on A to make A explicit. A valuation v on A* satisfies *an equality constraint $\langle G \equiv G' \rangle$ on A, written $v \models \langle G \equiv G' \rangle$, if $v(G)$ and $v(G')$ generate the same ideal. A* solution *of a constraint set (A, C) is a valuation on A that satisfies all constraints in C. If v is a solution of (A, C), we write $v \models (A, C)$, or simply $v \models C$. A template $a_1p_1 + \cdots + a_mp_m$ is a* GH template of g-degree τ under Γ *if $p_1, \ldots, p_m$ are GH polynomials of g-degree τ.*

We extend the definition of the remainder computation to operate on templates.

Definition 6. $\mathbf{Rem}^{\mathtt{par}}(A, f, p)$ *is a pair $(A', f - pq)$ where q is the most general template of degree $\mathbf{deg}(f) - \mathbf{deg}(p)$, the parameters of which are fresh; A' is the set of the parameters appearing in q. We write $\mathbf{Rem}^{\mathtt{par}}(A, \{ p_1, \ldots, p_m \}, p)$ for (A', G'), where $(A_i, r_i) = \mathbf{Rem}^{\mathtt{par}}(A, p_i, p)$ and $A' = \bigcup A_i$ and $G' = \{ r_1, \ldots, r_m \}$.*

For example, if the set of variables is $\{ x \}$, then $\mathbf{Rem}^{\mathtt{par}}(\emptyset, x^2, x+1) = (\{ a_1, a_2 \}, x^2 - (a_1x + a_2)(x+1))$; the most general template of degree $\mathbf{deg}(x^2) - \mathbf{deg}(x+1) = 1$ with variable x is $a_1x + a_2$. By expressing a remainder using a template, we can postpone the choice of a remainder operator to a later stage; for example, if we instantiate (a_1, a_2) with $(1, -1)$, then we have the standard remainder operator on $\mathbb{R}[x]$.

Algorithm 1 Inference of polynomial invariants.

1: **procedure** InvInf(c, d)
2: $g \leftarrow$ the most general template of degree d
3: $A_0 \leftarrow$ the set of the parameters occurring in g
4: $(A, G, C) \leftarrow [\![c]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathtt{par}}}(A_0, \{\, g \,\}, \emptyset)$
5: **return** $v(g)$ where v is a solution of $C \cup \{\, \langle G \equiv \{\, 0 \,\}\rangle \,\}$
6: **end procedure**

We recall the constraint generation algorithm proposed by Cachera et al. We write (A_i, G_i, C_i) for $[\![c_i]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathtt{par}}}(A, G, C)$ in each case of the following definition.

$$
\begin{aligned}
[\![\mathbf{skip}]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathtt{par}}}(A, G, C) &= (A, G, C) \\
[\![x{:=}p]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathtt{par}}}(A, G, C) &= (A, G[x := p], C) \\
[\![c_1; c_2]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathtt{par}}}(A, G, C) &= [\![c_1]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathtt{par}}}([\![c_2]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathtt{par}}}(A, G, C)) \\
[\![\mathbf{if}\ p = 0\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathtt{par}}}(A, G, C) &= (A_3, p \cdot G_2 \cup G_3, C_1 \cup C_2) \\
&\qquad \text{where} \quad (A_3, G_3) = \mathbf{Rem}^{\mathtt{par}}(A_1 \cup A_2, G_1, p) \\
[\![\mathbf{while}\ p \bowtie 0\ \mathbf{do}\ c_1]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathtt{par}}}(A, G, C) &= (A_1, G, C_1 \cup \{\langle G \equiv G_1 \rangle\})
\end{aligned}
$$

$[\![c]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathtt{par}}}(A, G, C)$ accumulates the generated parameters to A and the generated constraints to C. A is augmented by fresh parameters at the **if** statement where $\mathbf{Rem}^{\mathtt{par}}$ is called. At a **while** statement, $\langle G \equiv G_1 \rangle$ is added to the constraint set to express the loop-invariant condition.

Algorithm 1 solves the postcondition problem with the constraint-generating subprocedure $[\![c]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathtt{par}}}$. This algorithm, given a program c and degree d, returns a set of postconditions that can be expressed by an algebraic condition with degree d or lower. The algorithm generates the most general template g of degree d for the postcondition and applies $[\![c]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathtt{par}}}$ to g. For the returned set of polynomials G and the constraint set C, the algorithm computes a solution of $C \cup \langle G \equiv \{\, 0 \,\}\rangle$; the equality constraint $\langle G \equiv \{\, 0 \,\}\rangle$ states that $v(g) = 0$, where v is a model of the constraint set $C \cup \langle G \equiv \{\, 0 \,\}\rangle$, has to hold at the end of c regardless of the initial state.

This algorithm is proved to be sound: If $p \in \textsc{InvInf}(c, d)$, then $p = 0$ holds at the end of c for any initial states [3]. Completeness is not mentioned in their paper.

Example 7. We explain how $\textsc{InvInf}(c_{fall}, 3)$ works. The algorithm generates a degree-3 template $q(x, v, t, x_0, v_0, t_0, a, dt, g, \rho)$ over $\{\, x, v, t, x_0, v_0, t_0, a, dt, g, \rho \,\}$. The algorithm then generates the following constraints by $[\![c_{fall}]\!]^{\sharp \mathtt{cH}}_{\mathbf{Rem}^{\mathtt{par}}}$: $\langle \{\, q(x, v, t, x_0, v_0, t_0, a, dt, g, \rho) \,\}$ $\equiv \{\, q(x + v\,dt, v - g\,dt - \rho v\,dt, t + dt, x_0, v_0, t_0, a, dt, g, \rho) \,\}\rangle$ (from the body of the loop), $\langle \{\, q(x, v, t, x_0, v_0, t_0, a, dt, g, \rho) \,\} \equiv \{\, q(x_0, v_0, t_0, x_0, v_0, t_0, a, dt, g, \rho) \,\}\rangle$ (from the first statement of c_{fall}), and $\langle \{\, q(x_0, v_0, t_0, x_0, v_0, t_0, a, dt, g, \rho) \,\} \equiv \{\, 0 \,\}\rangle$. By solving these constraints with a solver for ideal membership problems [4] or with the heuristics proposed by Cachera et al. [3], and by applying the solution to $q(x, v, t, x_0, v_0, t_0, a, dt, g, \rho)$, we obtain p in Example 2.

Remark 1. The algorithm requires a solver for the constraints of the form $\langle G \equiv G' \rangle$. This is the problem of finding v that equates $\langle G \rangle$ and $\langle G' \rangle$; therefore, it can

Algorithm 2 Inference of polynomial invariants (homogeneous version.)

1: **procedure** $\textsc{InvInf}^{\mathtt{H}}(c,\, d,\, \Gamma,\, \tau)$
2: $\quad g \leftarrow$ the most general template of g-degree τ and degree d
3: $\quad A_0 \leftarrow$ the set of the parameters occurring in g
4: $\quad (A, G, C) \leftarrow [\![c]\!]^{\sharp\mathtt{cH}}_{\mathbf{Rem}^{\mathrm{par}},\Gamma}(A_0, \{\, g \,\}, \emptyset)$
5: $\quad$ **return** $v(g)$ where v is a solution of $C \cup \{\, \langle G \equiv \{\, 0 \,\}\rangle \,\}$
6: **end procedure**

be solved by repeatedly using a solver for the ideal membership problems [4]. To avoid high-cost computation, Cachera et al. proposed heuristics to solve an equality constraint.

6.2 Restriction to GH templates

We define a variation of the constraint generation algorithm in which we use only GH polynomial templates. The algorithm $[\![c]\!]^{\sharp\mathtt{cH}}_{\mathbf{Rem}^{\mathrm{par}},\Gamma}$ differs from $[\![c]\!]^{\sharp\mathtt{c}}_{\mathbf{Rem}^{\mathrm{par}}}$ in that it is parameterized also over Γ, not only over the remainder operation used in the algorithm. The remainder operator $\mathbf{Rem}^{\mathtt{parH}}_{\Gamma}(A, f, p)$ returns a pair $(A \cup A', f - pq)$ where q is the most general GH template with g-degree $\mathbf{gdeg}(f)\mathbf{gdeg}(p)^{-1}$, with degree $\mathbf{deg}(f) - \mathbf{deg}(p)$, and with fresh parameters; A' is the set of the parameters that appear in q. We again write (A_i, G_i, C_i) for $[\![c_i]\!]^{\sharp\mathtt{c}}_{\mathbf{Rem}^{\mathrm{par}}}(A, G, C)$ in each case of the following definition.

$$
\begin{aligned}
[\![\mathbf{skip}]\!]^{\sharp\mathtt{cH}}_{\mathbf{Rem}^{\mathrm{par}},\Gamma}(A,G,C) &= (A,G,C)\\
[\![x{:=}p]\!]^{\sharp\mathtt{cH}}_{\mathbf{Rem}^{\mathrm{par}},\Gamma}(A,G,C) &= (A, G[x := p], C)\\
[\![c_1; c_2]\!]^{\sharp\mathtt{cH}}_{\mathbf{Rem}^{\mathrm{par}},\Gamma}(A,G,C) &= [\![c_1]\!]^{\sharp\mathtt{cH}}_{\mathbf{Rem}^{\mathrm{par}},\Gamma}([\![c_2]\!]^{\sharp\mathtt{cH}}_{\mathbf{Rem}^{\mathrm{par}},\Gamma}(A,G,C))\\
[\![\mathbf{if}\ p = 0\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2]\!]^{\sharp\mathtt{cH}}_{\mathbf{Rem}^{\mathrm{par}},\Gamma}(A,G,C) &= (A_3, p\cdot G_2 \cup G_3, C_1 \cup C_2)\\
&\qquad \text{where } (A_3, G_3) = \mathbf{Rem}^{\mathtt{parH}}_{\Gamma}(A_1 \cup A_2, G_1, p)\\
[\![\mathbf{while}\ p \bowtie 0\ \mathbf{do}\ c_1]\!]^{\sharp\mathtt{cH}}_{\mathbf{Rem}^{\mathrm{par}},\Gamma}(A,G,C) &= (A_1, G, C_1 \cup \{\langle G \equiv G_1\rangle\})
\end{aligned}
$$

Algorithm 2 is the variant of Algorithm 1, in which we restrict a template to GH one.

The algorithm $\textsc{InvInf}^{\mathtt{H}}$ takes the input τ that specifies the g-degree of the invariant at the end of the program c. We have not obtained a theoretical result about which τ to be passed to $\textsc{InvInf}^{\mathtt{H}}$ so that it generates a good invariant. However, during the experiments in Section 7, we found that the following strategy often works: *Pass the g-degree of the monomial of interest.* For example, if we are interested in a property related to x, then pass $\Gamma(x)$ (i.e., L) to $\textsc{InvInf}^{\mathtt{H}}$ for the invariant $-gt^2 + gt_0^2 - 2tv + 2t_0v_0 + 2x - 2x_0 = 0$. How to help a user to find such "monomial of her interest" is left as an interesting future direction.

The revised version of the invariant inference algorithm is sound; at the point of writing, completeness of $\textsc{InvInf}^{\mathtt{H}}$ with respect to $\textsc{InvInf}$ is open despite the completeness of $[\![c]\!]^{\sharp\mathtt{H}}_{\mathbf{Rem},\Gamma}$ with respect to $[\![c]\!]^{\sharp}_{\mathbf{Rem}}$.

Theorem 3 (Soundness). *Suppose $\Gamma \vdash c$, $d \in \mathbb{N}$, and $\tau \in \mathbf{GDeg}$. Set P_1 to the set of polynomials that can be returned by $\textsc{InvInf}^{\mathtt{H}}(c, d, \tau)$; set P_2 to those by $\textsc{InvInf}(c, d)$. Then, $P_1 \subseteq P_2$.*

7 Experiment

We implemented Algorithm 2 and conducted experiments. Our implementation Fastind$_{dim}$ takes a program c, a maximum degree d of the template g in the algorithm, and a monomial w. It conducts type inference of c to generate Γ and calls $\textsc{InvInf}^{\mathtt{H}}(c, d, \Gamma, \mathbf{gdeg}_{\Gamma}(w))$. The type inference algorithm is implemented with OCaml; the other parts (e.g., a solver for ideal-equality constraints) are implemented with Mathematica.

The type inference module is based on the unification-based algorithm proposed by Kennedy [11,12] that computes the principal typing of a given program. We extended this algorithm to assign a g-degree to each occurrence of a constant symbol. We explain this extension using the following program $\texttt{sumpower}_d$: $(x, y, s) := (X{+}1, 0, \underline{1})$; **while** $x \neq 0$ **do if** $y = 0$ **then** $(x, y) := (x{-}1, x)$ **else** $(s, y) := (s + y^d, y - 1)$. Our definition of g-degrees, under any g-degree assignment, gives g-degree $1 \in \mathbf{GDeg}$ to a constant; therefore, the only g-degree assignment consistent with this program is $\{\, x \mapsto 1, y \mapsto 1, s \mapsto 1, X \mapsto 1 \,\}$, because X is added to 1, $X + 1$ is assigned to x, 0 is assigned to y, and 1 is assigned to s. This g-degree assignment is not useful for reducing the size of a template. Our implementation addresses this issue by treating a constant symbol as a variable; for $\texttt{sumpower}_d$, it assigns T^d to the underlined occurrence of 1 and T to the other occurrences of the constant symbols[7]. This g-degree assignment indeed produces a smaller template.

To demonstrate the merit of our approach, we applied this implementation to the benchmark used in the experiment by Cachera et al. [3] and compared our result with that of their implementation, which is called Fastind. The entire experiment is conducted on a MacBook Air 13-inch Mid 2013 model with a 1.7 GHz Intel Core i7 (with two cores, each of which has 256 KB of L2 cache) and 8 GB of RAM (1600 MHz DDR3). The modules written in OCaml are compiled with `ocamlopt`. The version of OCaml is 4.02.1. The version of Mathematica is 10.0.1.0. We refer the reader to [3, 14, 15] for detailed descriptions of each program in the benchmark. They contain a nested loop with a conditional branch (e.g., `dijkstra`), a sequential composition of loops (e.g., `divbin`), and nonlinear expressions (e.g., `petter(n)`.) We generate a nonlinear invariant in each program.

Table 1 shows the result. The column deg shows the degree of the generated polynomial; t_{sol} shows the time spent by the ideal-equality solver (ms); $\#m$ shows the number of monomials in the generated template; t_{inf} shows the time spent by the dimension-type inference algorithm (ms); and $t_{inf} + t_{sol}$ shows the sum of $t_{inf} + t_{sol}$. By comparing $\#m$ for Fastind with that for Fastind$_{dim}$, we can discuss the effect of the use of GH polynomials to the size of templates; comparison of t_{sol} for Fastind with that for Fastind$_{dim}$ suggests the effect to the constraint reduction phase; comparison of t_{sol} for Fastind with $t_{inf} + t_{sol}$ for Fastind$_{dim}$ suggests the overhead incurred by g-degree inference.

[7] Our implementation generates the set of base degrees B automatically.

Name		Fastind			Fastind$_{dim}$			
	deg	t_{sol}	$\#m$	deg	t_{inf}	t_{sol}	$t_{inf}+t_{sol}$	$\#m$
dijkstra	2	9.29	21	2	0.456	8.83	9.29	21
divbin	2	0.674	21	2	0.388	0.362	0.750	8
freire1	2	0.267	10	2	0.252	0.258	0.510	10
freire2	3	2.51	35	3	0.463	2.60	3.06	35
cohencu	3	1.74	35	3	0.434	0.668	1.10	20
fermat	2	0.669	21	2	0.583	0.669	1.25	21
wensley	2	104	21	2	0.436	28.5	28.9	9
euclidex	2	1.85	45	3	1.55	1.39	2.94	36
lcm	2	0.811	28	2	0.513	0.538	1.05	21
prod4	3	31.6	84	3	0.149	2.78	2.93	35
knuth	3	137	220	3	4.59	136	141	220
mannadiv	2	0.749	21	3	0.515	0.700	1.22	18
petter1	2	0.132	6	2	0.200	0.132	0.332	6
petter2	3	0.520	20	3	0.226	0.278	0.504	6
petter3	4	1.56	35	4	0.226	0.279	0.505	7
petter4	5	7.15	56	5	0.240	0.441	0.681	8
petter5	6	17.2	84	6	0.228	0.326	0.554	9
petter10	11	485	364	11	0.225	0.354	0.579	14
sumpower1	3	2.20	35	3	0.489	2.31	2.80	35
sumpower5	7	670	330	7	0.469	89.1	89.6	140

Table 1. Experimental result.

Discussion The size of the templates, measured as the number of monomials ($\#m$), was reduced in 13 programs out of 20 by using GH polynomials. The value of t_{sol} decreased for these 13 programs; it is almost the same for the other programs. $\#m$ did not decrease for the other 7 programs because the extension of the type inference procedure mentioned above introduced useless auxiliary variables. We expect that we can eliminate such variables by using more elaborate program analysis.

By comparing t_{sol} for Fastind and $t_{inf}+t_{sol}$ for Fastind$_{dim}$, we can observe that the inference of g-degree assignment sometimes incurred an overhead for the entire execution time if the template generated by Fastind was small enough and therefore Fastind was already efficient. However, this overhead is compensated for in the programs for which Fastind requires more computation time.

To summarize, our current approach is especially effective for a program for which (1) the existing invariant-synthesis algorithm is less efficient owning to the large size of the template and (2) we can infer a nontrivial g-degree assignment. We expect that our approach will be effective for a wider range of programs if we find a more competent g-degree inference algorithm.

8 Related work

The template-based algebraic invariant synthesis proposed to date [3, 16] have focused on how to reduce the problem to constraint solving and how to solve the generated constraints efficiently; strategies for generating a template have not been the main issue. A popular strategy for template synthesis is to iteratively increase the degree of a template. This strategy suffers from an increase in the size of a template in the iterations with high degree.

Our claim is that prior analysis of a program effectively reduces the size of a template; we used the dimension type system for this purpose in this paper inspired by the principle of quantity dimensions in the area of physics. Of course, there is a tradeoff between the cost of the analysis and its effect in the template-size reduction; our experiments suggest that the cost of dimension type inference is reasonable.

Semialgebraic invariants (i.e., invariants written using *inequalities* on polynomials) are often useful for program verification. The template-based approach is also popular in semialgebraic invariant synthesis. One popular strategy in template-based semialgebraic invariant synthesis is to reduce this problem to one of semidefinite programming, for which many efficient solvers are widely available.

As of this writing, it is an open problem whether our idea regarding GH polynomials also applies to semialgebraic invariant synthesis; for physically meaningful programs, at least, we guess that it is reasonable to use GH polynomials because of the success of the quantity dimension principle in the area of physics. A possible approach to this problem would be to investigate the relationship between GH polynomials and Stengle's Postivstellensatz [18], which is the theoretical foundation of the semidefinite-programming approach mentioned above. There is a homogeneous version of Positivstellensatz [8, Theorem II.2]; because the notion of homogeneity considered there is equivalent to generalized homogeneity introduced in this paper, we conjecture that this theorem provides a theoretical foundation of an approach to semialgebraic invariant synthesis using GH polynomials.

Although the application of the quantity dimension principle to program verification is novel, this principle has been a handy tool for discovering hidden knowledge about a physical system. A well-known example in the field of hydrodynamics is the motion of a fluid in a pipe [1]. One fundamental result in this regard is that of Buckingham [2], who stated that *any physically meaningful relationship among n quantities can be rewritten as one among $n - r$ independent dimensionless quantities, where r is the number of the quantities of the base dimension.* Investigating the implications of this theorem in the context of our work is an important direction for future work.

The term "generalized homogeneity" appears in various areas; according to Hankey et al. [10], a function $f(x_1, \ldots, x_n)$ is said to be generalized homogeneous if there are $a_1, \ldots, a_n$ and a_f such that, for any positive λ, $f(\lambda^{a_1}, \ldots, \lambda^{a_n}) = \lambda^{a_f} f(x_1, \ldots, x_n)$. Barenblatt [1] points out that the essence quantity dimension principle is generalized homogeneity. Although we believe our GH polynomials is

related to the standard definition, we have not fully investigated the relationship at the time of writing.

9 Conclusion

We presented a technique to reduce the size of the template used in template-based invariant-synthesis algorithms. Our technique is based on the finding that, if an algebraic invariant of a program c exists, then there is a GH invariant of c; hence, we can reduce the size of a template by synthesizing only a GH polynomial. We presented the theoretical development as a modification of the framework proposed by Cachera et al. and empirically confirmed the effect of our approach using the benchmark used by Cachera et al. Although we used the framework of Cachera et al. as a baseline, we believe that we can apply our idea to the other template-based methods [3, 7, 13, 15–17].

Our motivation for the current work is safety verification of hybrid systems, in which the template method is a popular strategy. For example, Gulwani et al. [9] proposed a method of reducing the safety condition of a hybrid system to constraints on the parameters of a template by using Lie derivatives. We expect our idea to be useful for expediting these verification procedures.

We are also interested in applying our idea to decision procedures and satisfiability modulo theories (SMT) solvers. Support of nonlinear predicates is an emerging trend in many SMT solvers (e.g., Z3 [6]). Dai et al. [5] proposed an algorithm for generating a semialgebraic Craig interpolant using semidefinite programming [5]. Application of our approach to these method is an interesting direction for future work.

References

1. G. I. Barenblatt. *Scaling, self-similarity, and intermediate asymptotics: dimensional analysis and intermediate asymptotics*, volume 14. Cambridge University Press, 1996.
2. E. Buckingham. On physically similar systems; illustrations of the use of dimensional equations. *Phys. Rev.*, 4:345–376, Oct 1914.
3. D. Cachera, T. P. Jensen, A. Jobin, and F. Kirchner. Inference of polynomial invariants for imperative programs: A farewell to Gröbner bases. *Sci. Comput. Program.*, 93:89–109, 2014.
4. D. A. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
5. L. Dai, B. Xia, and N. Zhan. Generating non-linear interpolants by semidefinite programming. In N. Sharygina and H. Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 364–380. Springer, 2013.

6. L. M. de Moura and N. Bjørner. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, 2008.
7. P. Garg, C. Löding, P. Madhusudan, and D. Neider. ICE: A robust framework for learning invariants. In A. Biere and R. Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 69–87. Springer, 2014.
8. L. Gonzalez-Vega and H. Lombardi. Smooth parametrizations for several cases of the Positivstellensatz. *Mathematische Zeitschrift*, 225(3):427–451, 1997.
9. S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In A. Gupta and S. Malik, editors, *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, volume 5123 of *Lecture Notes in Computer Science*, pages 190–203. Springer, 2008.
10. A. Hankey and H. E. Stanley. Systematic application of generalized homogeneous functions to static scaling, dynamic scaling, and universality. *Physical Review B*, 6(9):3515, 1972.
11. A. Kennedy. Dimension types. In D. Sannella, editor, *Programming Languages and Systems - ESOP'94, 5th European Symposium on Programming, Edinburgh, U.K., April 11-13, 1994, Proceedings*, volume 788 of *Lecture Notes in Computer Science*, pages 348–362. Springer, 1994.
12. A. Kennedy. *Programming Languages and Dimensions*. PhD thesis, St. Catharine' s College, Mar. 1996.
13. M. Müller-Olm and H. Seidl. Computing polynomial program invariants. *Inf. Process. Lett.*, 91(5):233–244, 2004.
14. E. Rodríguez-Carbonell. Some programs that need polynomial invariants in order to be verified. `http://www.cs.upc.edu/~erodri/webpage/polynomial_invariants/list.html` (Accessed on January 25th, 2016).
15. E. Rodríguez-Carbonell and D. Kapur. Generating all polynomial invariants in simple loops. *J. Symb. Comput.*, 42(4):443–476, 2007.
16. S. Sankaranarayanan, H. Sipma, and Z. Manna. Non-linear loop invariant generation using Gröbner bases. In N. D. Jones and X. Leroy, editors, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, pages 318–329. ACM, 2004.
17. F. Somenzi and A. R. Bradley. IC3: where monolithic and incremental meet. In P. Bjesse and A. Slobodová, editors, *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, pages 3–8. FMCAD Inc., 2011.
18. G. Stengle. A nullstellensatz and a positivstellensatz in semialgebraic geometry. *Mathematische Annalen*, 207(2):87–97, 1974.

A Proof of Theorem 3

To prove Theorem 3, we need to define *renaming* of parameters and constraints.

Definition 7. *For an injection $\iota : A \to A'$, we write $\iota : (A, G, C) \preceq (A', G', C')$ if $G' = \iota^*(G)$ and $C' = \iota^*(C)$ where ι^* maps $a' \in \iota(A)$ to $\iota^{-1}(a')$ and $a' \in \iota(A' \backslash \iota(A))$ to 0.*

The injection ι gives a renaming of parameters. The relation $\iota : (A, G, C) \preceq (A', G', C')$ reads G and C are obtained from G' and C' by renaming the parameters in $\iota(A)$ using ι and substituting 0 to those not in $\iota(A)$.

Lemma 2. *If $\iota : (A, G, C) \preceq (A', G', C')$, then there exists κ such that (1) $\kappa : [\![c]\!]^{\sharp \mathtt{cH}}_{\mathbf{Rem}^{\mathrm{par}}, \Gamma}(A, G, C) \preceq [\![c]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathrm{par}}}(A', G', C')$ and (2) κ is an extension of ι.*

Proof. Induction on the structure of c. □

Proof of Theorem 3 Let $g \in T(A_0)$ be the most general template of generalized degree τ and degree d, and $g' \in T(A'_0)$ be the most general template of degree d. Without loss of generality, we can assume $A_0 \subseteq A'_0$ and $g' = g + g_1$ for some $g_1 \in T(A'_0 \backslash A_0)$. Let $(A, G, C) = [\![c]\!]^{\sharp \mathtt{cH}}_{\mathbf{Rem}^{\mathrm{par}}, \Gamma}(A_0, \{ g \}, \emptyset)$ and $(A', G', C') = [\![c]\!]^{\sharp \mathtt{c}}_{\mathbf{Rem}^{\mathrm{par}}}(A'_0, \{ g' \}, \emptyset)$. Then, from Lemma 2, there exists κ such that $\kappa : (A, G, C) \preceq (A', G', C')$ and κ is an extension of the inclusion mapping $\iota : A_0 \to A'_0$. Suppose $v(g)$ is a result of $\textsc{InvInf}^{\mathtt{H}}(c, d, \tau)$ where v is a solution to $C \cup \{ \langle G \equiv \{ 0 \} \rangle \}$. Define a valuation v' on A' by

$$v'(a') = \begin{cases} v(a) & a' = \kappa(a) \text{ for some } a \in A \\ 0 & \text{Otherwise.} \end{cases}$$

Then, $v'(g') = v'(g + g_1) = v'(g)$; the second equation holds because $v'(a')$ is constantly 0 on any $a' \in A' \backslash A$. All the parameters in g are in A_0 and κ is an identity on A_0. Therefore, $v'(g) = v(g)$. It suffices to show that $v' \models C' \cup \{ \langle G' \equiv \{ 0 \} \rangle \}$, which indeed holds from the definition of v' since $v \models C \cup \{ \langle G \equiv \{ 0 \} \rangle \}$ and C and G are renaming of C' and G'. □