Interpolation-based height analysis for improving a recurrence solver

Manuel Montenegro[1], Olha Shkaravska[2], Marko van Eekelen[2&3], and Ricardo Peña[1]

[1] Universidad Complutense Madrid
[2] Radboud University Nijmegen
[3] Open University of the Netherlands
montenegro@fdi.ucm.es,{shkarav,marko}@cs.ru.nl,ricardo@sip.ucm.es

Abstract. The COSTA system infers resource consumption bounds from Java bytecode using an internal recurrence solver PUBS. This paper suggests an improvement of the COSTA system, such that it can solve a larger number of recurrences. The idea is to replace one of its static analyses, the ranking function analysis, by another kind of analysis, height analysis, in such a way that polynomial bounds of any degree may be inferred instead of just linear expressions. The work can be seen as an application of some polynomial interpolation techniques used by some of the authors in prior analyses. Finding a way to choose proper test nodes is the key to the solution presented in this paper.

1 Introduction

The application of resource analysis techniques to actual programs frequently leads to the generation of cost recurrence equations which must be solved, i.e. expressed in a closed form in order to be useful. There have been many attempts to solve this kind of equations in an automatic way. Most of the tools doing this are restricted to specific recurrence families, frequently on only one variable.

A system proved successful in generating and solving multivariate recurrence equations is COSTA [1], together with its associated recurrence solving subsystem PUBS [2]. COSTA is given a program text written in Java bytecode, and the kind of resource to be analysed (number of instructions executed, memory consumption, among others), and in many cases it is able to obtain symbolic upper bounds for these runtime figures. The bounds are expressed as multivariate functions on the sizes of the input arguments of the method being analysed.

Internally, the system contains many different static analyses which we briefly summarise in Sec. 2. In this paper, we focus on one of them: the *ranking function synthesis* done in one of the steps. Its aim is to obtain an upper bound on the number of unfoldings the recurrence equations must undergo to reach a base

This work was partly funded by the EU Artemis Joint Undertaking in the CHARTER project, grant-nr. 100039, and by Spanish FPU grant AP2006-02154 and projects TIN2008-06622-C03-01 (STAMP), S2009/TIC-1465 (PROMETIDOS).

case, i.e. to reach a case with does not admit more unfoldings. This number corresponds with the maximum height of the so-called *evaluation trees*. As we will see, this ranking function is crucial in the computation of the resource upper bound. The limitation of this step is that the ranking function must be a *linear expression* on the argument sizes. The method of Podelski and Rybalchenko [20], which is known to be complete for linear ranking functions, is used for this purpose.

In the past, we have used polynomial interpolation techniques in different types of size analysis [21], and also in synthesising polynomial ranking functions for Java loops [24]. The main idea there was to synthesise polynomials by evaluating program fragments, or particular term rewriting systems, in some specific test points arranged in a so-called NCA configuration (Node Configuration A, [8]). By adopting this configuration the number of test points needed is a precise function of the polynomial degree and the number of variables.

We apply the same idea here by evaluating recurrence equations at well-chosen points and then interpolating a multivariate polynomial of known degree passing through those points. As we are only interested in the maximum height of the recurrence evaluation trees (the rest of the recurrence solving process is already done by PUBS), we first write a derived recurrence defining this height. The polynomial obtained would be then an upper bound of the height of any evaluation tree. The degree must be guessed, as happens in other techniques inferring arbitrary polynomials [18, 14], and additionally the interpolated polynomial must be proved correct w.r.t. the given recurrence equations. We will show that the process can be performed fully algorithmically by using appropriate tools. In the end we have extended the power of COSTA by allowing the solution of more general recurrences, specifically those admitting a polynomial upper bound on the height of the evaluation trees.

The plan of the paper is as follows: after this introduction, in Sec. 2 we summarise the main components of the COSTA-PUBS system and the process it follows for recurrence solving; in Sec. 3 we explain the main steps of our inference process by using a small running example; sections 4, 5, and 6 contain the technical details of respectively what we mean by evaluating a non-deterministic recurrence, how to choose appropriate test points, and how to prove the obtained polynomial correct; finally, Sec. 7 surveys the related work and draws some conclusions.

2 A Broad Overview of COSTA and PUBS

The COSTA System is based on the classical approach to resource analysis, due to Wegbreit [27]. Given a Java bytecode program and a cost model, COSTA generates, in a first phase, a set of equations specifying the cost of the program as a function on the size of its input. The meaning of the *size of the input* depends on the type of the latter: the size of an integer is its value, whereas the size of an array is its number of elements. Finally, the size of an arbitrary object is the longest reachable pointer path that stems from that object. As an example, we

show below a code fragment with a recurrence relation specifying its memory consumption [1]:

```
while (n > 0) {
   int[] arr = new int[n];
   n--;
}
```

$$\begin{array}{ll} T(n) = 0 & \{n \leq 0\} \\ T(n) = 4n + T(n-1) & \{n > 0\} \end{array} \quad (1)$$

Although a recurrence relation captures precisely the cost of a program, an equivalent expression without recursion (*closed form*) gives a more intuitive idea about these costs from the programmer's point-of-view. For instance, the recurrence shown above admits the following closed-form: $T(n) = 2n^2 + 2n$. So, the second phase of COSTA consists of the computation of a closed form for the previously generated set of equations. In contrast to already existing tools for solving recurrence relations [5], COSTA provides its own recurrence solver, PUBS [2]. The main reason is that, in practice, the set of equations capturing the cost of a program are not recurrence relations, but belong to the broader class of *Cost Relation Systems* (CRS), which are defined as follows:

Definition 1 (Adapted from [2]). *A cost relation system is a finite set of equations of the form:*

$$T(\bar{x}) = \texttt{exp} + \sum_{j=1}^{l} T(\bar{y}_j) \qquad \psi$$

where $l \geq 0$, `exp` *is a basic cost expression (i.e. does not refer to other cost functions such as T) and ψ is a set of linear relations on $\bar{x} \cup \{\bar{y}_j\}_{j=1..l} \cup \text{vars}(\texttt{exp})$ specifying the conditions under which the equation can be applied.*

The formal definition of basic cost expressions is not relevant to this paper, and it shall be omitted here (see [2] for details). Our definition of CRS differs slightly from that of [2] in the sense that, in the latter, there could be occurrences of cost symbols different from T in the right-hand side of the equations. However, as pointed out in that paper, it is possible to unfold the definitions of these symbols in order to obtain equations as in the definition above, so we assume that this transformation has already been done. The main difference between CRS and regular recurrence relations is *non-determinism*: when evaluating $T(\bar{v})$ for some concrete values, we might be able to apply more than one equation. Moreover, the arguments $T(\bar{y}_j)$ occurring in the recursive calls of an equation may not be uniquely determined by the guards, as in, for example, $T(x) = 1 + T(x') \ \{0 \leq x' < x\}$. Thus the evaluation of $T(\bar{v})$ may give rise to different results and hence a CRS defines a relation instead of a function.

The PUBS approach for computing a closed form upper-bound is based on the notion of an *evaluation tree* (ET), which is the structure describing the

[1] Here we follow the simplified model of COSTA, where it is assumed that e.g. an integer array of length n allocates $4n$ bytes. In practice allocation of arrays is compiler-dependent and typically assumes packing and optimisations.

evaluation of $T(\bar{v})$ for some values $\bar{v}$. Each node in the ET contains the cost of its basic cost expression `exp` applied to the $\bar{v}$, and its children correspond to the recursive calls $T(\bar{v}_j)$, where $j \in \{1..m\}$. Notice that the evaluation of $T(\bar{v})$ may be described by several different ETs, because of non-determinism. The computation of an upper bound to $T(\bar{x})$ amounts to finding upper-bounds to:

- The number of base and recursive nodes of every possible ET, respectively denoted by $\mathrm{nb}(\bar{x})$ and $\mathrm{nr}(\bar{x})$.
- The value of the basic cost expressions occurring in the base and recursive nodes of every possible ET, respectively denoted by $\mathrm{cb}(\bar{x})$ and $\mathrm{cr}(\bar{x})$.

Given these, an upper bound $T^+(\bar{x})$ is computed as follows:

$$T^+(\bar{x}) = \mathrm{nb}(\bar{x}) * \mathrm{cb}(\bar{x}) + \mathrm{nr}(\bar{x}) * \mathrm{cr}(\bar{x}) \tag{2}$$

The computation of $\mathrm{cb}(\bar{x})$ and $\mathrm{cr}(\bar{x})$ is beyond the scope of this paper. With regard to the number of nodes in the ET (nb and nr), these functions are approximated (see [2]) from the ET's branching factor b and maximal height $h(\bar{x})$ as follows:

$$\mathrm{nb}(\bar{x}) = b^{h(\bar{x})} \qquad \mathrm{nr}(\bar{x}) = \begin{cases} h(\bar{x}) & \text{if } b = 1 \\ \frac{b^{h(\bar{x})}-1}{b-1} & \text{if } b > 1 \end{cases}$$

The h function stands for *the length of the maximal call chain* (without including the base case) that stems from the root of the ET. For example, with the CRSs shown in (1) the only (and hence the longest) chain reachable from $T(n)$ is the following:

$$\underbrace{T(n) \to T(n-1) \to T(n-2) \to \ldots \to T(1)}_{h(n)=n} \to T(0)$$

In order to compute $h(\bar{x})$, PUBS derives a ranking function for T by applying the method of Podelski and Rybalchenko [20]. This method is targeted towards the termination of loops whose conditions and update statements are given by linear expressions, and it is able to synthesize linear ranking functions by constructing a system of linear inequalities over rationals. This system characterizes the existence of such ranking functions: the system is solvable if and only if there exists a linear ranking function for the given loop. Thus this method is complete for linear ranking functions. Unfortunately, it fails when the ranking function does not depend linearly on the arguments $\bar{x}$, as the following example shows:

Example 1. Let us assume the following loop:

```
while(x > 0 || y > 0) {
   byte[] b = new byte[x];
   if (y == 0) { x--; y=x; } else { y--; }
}
```

If we consider memory consumption, COSTA would obtain the following CRS:

$$\begin{array}{ll} T(x,y) = \mathsf{nat}(x) & \{x = 0, y = 0\} \\ T(x,y) = \mathsf{nat}(x) + T(x-1, x-1) & \{x > 0, y = 0\} \\ T(x,y) = \mathsf{nat}(x) + T(x, y-1) & \{x \geq 0, y > 0\} \end{array}$$

where $\mathsf{nat}(x)$ abbreviates $\max\{0, x\}$. This CRS cannot be solved by PUBS, since the length of the longest call chain depends on (x, y) in a non-linear way.

$$\underbrace{T(x,y) \to T(x,y-1) \to \ldots \to T(x,0) \to T(x-1,x-1) \to \ldots \to T(0,0)}_{h(x,y)=\frac{1}{2}x^2+\frac{1}{2}x+y} \to T(0)$$

3 Interpolation-based call chain height analysis

COSTA derives cost recurrence equations from Java bytecode. From these CRSs COSTA derives ranking functions. Ranking functions are used to bound the height of the evaluation trees. In COSTA such ranking functions are limited to linear expressions. It is our goal to improve the approximation of the height of the evaluation trees by considering general polynomial expressions. Using polynomial interpolation we aim to derive those polynomial expressions by analysing the call chain directly. To this purpose we first derive equations for the height of the call chain from the COSTA cost relation equations. Then, we derive upper bounds by polynomial interpolation.

Firstly, we must determine the function to be interpolated. We transform a given original CRS in order to obtain a recursive definition modeling the height $T_h(\bar{x})$ of the evaluation tree:

- for non-recursive equations of the form:

$$T(\bar{x}) = \mathtt{exp} \quad \psi \qquad \text{we get:} \qquad T_h(\bar{x}) = 0 \quad \psi$$

- for recursive equations of the form:

$$T(\bar{x}) = \mathtt{exp} + \sum_{i=1}^{m} T(\overline{y_i}) \quad \psi \qquad \text{we get:} \qquad T_h(\bar{x}) = 1 + \max_{i=1\ldots m} \{T_h(\overline{y_i})\} \quad \psi$$

The function T_h is multivalued: $T_h(\bar{x})$ is the collection of the lengths of all the call chains rooting in $\bar{x}$. Recall, that the function $h(\bar{x})$ is the maximum length of all possible call chains from $\bar{x}$, therefore $h(\bar{x})$ is the strict upper bound for $T_h(\bar{x})$.

Next, after we have the recurrence relations for $T_h(\bar{x})$, our aim is to find a polynomial $T_h^+(\bar{x})$ such that $T_h^+(\bar{x}) \geq h(\bar{x})$. If $h(\bar{x})$ is a polynomial, then $T_h^+(\bar{x})$ may be found by the standard interpolation. For instance, this is the case for the Example 1. We will consider it in more detail later in this section. If $h(\bar{x})$ is not a polynomial then the standard interpolation must be adjusted. In both

cases the idea is to generate a candidate for $T_h^+(\bar{x})$ based on its values in some finite collection of *nodes*. Then the candidate is checked. Roughly, checking is done by substituting it into the recurrence relation for $T_h(\bar{x})$.

In this paper we consider in more detail the issues related to the generation of a candidate for an upper bound. The most important issue here is to find a collection of test nodes such that the corresponding interpolation problem has a solution (and it must be unique). There are many ways to obtain test nodes. In our work we use the gradient-based approach, which is described in Sec. 5.

Now we briefly recapitulate prerequisites from the classical interpolation theory that are necessary to understand the major challenge: finding test nodes. A detailed survey of multivariate interpolation can be found e.g. in paper [23].

One-variable interpolation problem is well-known. For instance, to define a line p (i.e. a linear function) one needs to know two points on the plane: some $\big(x_1, p(x_1)\big)$ and $\big(x_2, p(x_2)\big)$. In general, to define a polynomial of degree d of one variable one needs to know its value in $d+1$ different points. This is due to the fact that this polynomial has $d+1$ coefficients, which solves the linear system generated by conditions $a_1 \, + \, a_2 \, z_i \, + \, \ldots \, + \, a_{d+1} \, z_i^d = \, p(z_i)$, where $1 \leq i \leq d+1$. If all the points z_i are different, then the determinant of the matrix of this system is non-zero Vandermonde determinant. This provides the existence and uniqueness of the solutions $a_1, \ldots, a_{d+1}$.

There exists a similar condition that assures non-singularity of *multivariate* Vandermonde matrix, and therefore existence and uniqueness of the solution of the underlying interpolation problem. Recall that a polynomial of degree d and dimension s (the number of variables) has $N_d^s = \binom{d+s}{s}$ coefficients. Let a set of values f_i of a real function f be given. A set $W = \{\bar{w}_i \, : \, i = 1, \ldots, N_d^s\}$ of points in a real s-dimensional space forms the set of *interpolation nodes* if there is a unique polynomial $p(\bar{z}) = \Sigma_{0 \leq |j| \leq d} a_j \bar{z}^j$ with the total degree d with the property $p(\bar{w}_i) = f_i$, where $1 \leq i \leq N_d^s$. In this case one says that the polynomial p *interpolates the function* f at the nodes $\bar{w}_i$.

The condition under which there exists a unique polynomial that interpolates *multivariate* data is not trivial. This condition on W is geometrical: it describes a configuration, called *Node Configuration A* (**NCA**) [8], in which the points from W should be placed in $\mathbb{R}^s$. The multivariate Vandermonde determinant computed from such points is non-zero. Thus, the corresponding system of linear equations w.r.t. the polynomial's coefficients has a unique solution. For a two-dimensional polynomial of degree d, the condition on the nodes that guarantees a unique polynomial interpolation is as follows:

N_d^2 nodes forming a set $W \subset \mathbb{R}^2$ lie in a 2-dimensional NCA if there exist lines $\gamma_1, \ldots, \gamma_{d+1}$ in the space $\mathbb{R}^2$, such that $d+1$ nodes of W lie on γ_{d+1} and d nodes of W lie on $\gamma_d \backslash \gamma_{d+1}$, ..., and finally 1 node of W lies on $\gamma_1 \backslash (\gamma_2 \cup \ldots \cup \gamma_{d+1})$.

A simple example of NCA is given by a rectangular grid: there are $d+1$ parallel lines, such that some $d+1$ points lie on one line, d points lie on another one, $d-1$ points belong on some third line, etc.

The *NCA configuration for s variables (s-dimensional space)* is defined inductively on s [8]. Let $\{\bar{z}_1, \ldots, \bar{z}_{N^s_d}\}$ be a set of distinct points in $\mathbb{R}^s$ such that there exist $d+1$ hyperplanes K^s_j, $0 \le j \le d$ with

$\bar{z}_{N^s_{d-1}+1}, \ldots, \bar{z}_{N^s_d} \in K^d_s$

$\bar{z}_{N^s_{j-1}+1}, \ldots, \bar{z}_{N^s_j} \in K^s_j \setminus \{K^s_{j+1} \cup \ldots \cup K^s_d\}$, for $0 \le j \le d-1$

and each of set of points $\bar{z}_{N^s_{j-1}+1}, \ldots, \bar{z}_{N^s_j}$, $0 \le j \le s$, considered as points in $\mathbb{R}^{s-1}$ satisfies **NCA** in $\mathbb{R}^{s-1}$.

3.1 Application of the approach to an example

Recall the recurrence relation in the Example 1. We have to find an interpolating polynomial of degree 2 for the corresponding call-tree-height function:

$$\begin{array}{ll} T_h(x,y) = 0 & \{x = 0, y = 0\} \\ T_h(x,y) = 1 + T_h(x-1, x-1) & \{x \ge 1, y = 0\} \\ T_h(x,y) = 1 + T_h(x, y-1) & \{x \ge 0, y \ge 1\} \end{array}$$

Evaluating T_h in 7 points forming an NCA yields the following table:

x	y	$T_h(x,y)$
0	0	$T_h(0,0) = 0$
1	0	$T_h(1,0) = 1 + T_h(0,0) = 1$
1	1	$T_h(1,1) = 1 + T_h(1,0) = 1 + 1 = 2$
1	2	$T_h(1,2) = 1 + T_h(1,1) = 3$
0	1	$T_h(0,1) = 1 + T_h(0,0) = 1$
0	2	$T_h(0,2) = 1 + T_h(0,1) = 1 + 1 = 2$
2	0	$T_h(2,0) = 1 + T_h(1,1) = 1 + 2 = 3$

By solving the resulting system we get the polynomial $T^+_h(x,y) = \frac{1}{2}x^2 + \frac{1}{2}x + y$. It is a routine to check that it fits the recurrence relations. We compute the following closed form expression for T^+ by using (2):

$$T^+(x,y) = \mathsf{nat}(\frac{1}{2}x^2 + \frac{1}{2}x + y) * \mathsf{nat}(x) + \mathsf{nat}(x)$$

and taking into account that the branching factor in this case is $b = 1$. □

3.2 When standard interpolation needs adjustment

When $h(\bar{x})$ is not a polynomial, but still, bounded by some polynomial, the standard interpolation must be adjusted. This is a topic of the ongoing research.

We illustrate the problem with a small example. Let us need to find a linear bound $p(x) \ge f(x)$ for a function $f(x)$ knowing that $f(0) = 1$ and $f(1) = 3$. If we assume that $p(x) = ax + b = f(x)$, we obtain from the interpolation data that $a \cdot 0 + b = 1$ and $a \cdot 1 + b = 3$. From this follows that $b = 1$ and $a = 3 - b = 2$,

and therefore $p(x) = 2x + 1$. But it can be that $f(x) = 6$ for all $x \geq 2$. The obtained polynomial is not an upper bound on f, since $f(2) > p(2)$, therefore checking fails. With the classical interpolation-based approach one picks up the third point, say, $x = 2$ and finds a quadratic bound, which in our case means a significant overestimation! However, instead of solving the interpolation problem, one can solve an optimisation problem. One takes more nodes and solve linear *inequations* $p(\bar{x}_i) \geq f_i$, e.g. trying to minimise the leading coefficients of p. In the example we solve $b \geq 1$, $a + b \geq 3$, $2a + b \geq 6$ and $a \geq 0$ with $g(b, a) = a \to \min$. Using linear optimisation we obtain the minimum for $g(b, a) = a$ in $(6, 0)$. From this it follows that $p(x) = b = 6$, which is the correct bound.

The question whether replacing interpolation problem with optimisation problem works well for more complex (multivariate) cases is open. In particular, one needs to find out with how many additional points, which objective functions, which additional inequations will provide the best results.

3.3 Towards non-deterministic cost relation systems

When the CRS defining T_h is deterministic (as in our previous examples) it defines a *single-valued* function whose evaluation can be done in the usual way, e.g. by unfolding. If we want to compute a upper-bound to $h(\bar{x})$ we just choose a set of points in the domain of T_h lying in a NCA configuration, and evaluate T_h in these points, as shown in the example above.

However, the computation of an upper-bound in non-deterministic CRSs is far more involved. The main reason is that we cannot just choose a point $\bar{x}$ and obtain all the possible results of the evaluation of $T_h(\bar{x})$, since there could be infinitely many ETs resulting from it. In the next two sections we explain how to obtain test nodes in which the value of T_h is known, and how to perform a search on those nodes in which interpolation is more likely to result in a correct upper-bound (gradient-based approach).

4 Evaluation of Cost Relation Systems

Before dealing with the evaluation of a CRS we have to define its semantics. For the sake of simplicity, we consider only CRSs with a single recursive call (the extension to CRSs with several calls is straightforward):

$$\begin{array}{ll} T_h(\bar{x}) = 0 & \psi_b(\bar{x}) \\ T_h(\bar{x}) = 1 + T_h(\bar{x}') & \psi_r(\bar{x}, \bar{x}') \end{array} \tag{3}$$

Because of the above mentioned non-determinism, these equations denote a *relation* $T_h \subseteq \mathbb{N}^s \times \mathbb{N}_\infty$, (where $\mathbb{N}_\infty = \mathbb{N} \cup \{+\infty\}$) rather than a function. The pair $(\bar{x}, n)$ belongs to T_h iff n is a result of the evaluation of $T_h(\bar{x})$. The intuitive meaning of $(\bar{x}, +\infty)$ being in T_h is that the evaluation of $T_h(\bar{x})$ does not terminate (i.e. may lead to an infinite call chain). The ordering $\leq$ on natural numbers and the $+$ operator is extended to $\mathbb{N}_\infty$ as usual. Given these conventions, the following definition specifies the semantics of the set of equations in (3).

Definition 2. *The relation T_h defined by the CRS in (3) is the greatest fixed point of the function $F : \mathcal{P}(\mathbb{N}^s \times \mathbb{N}_\infty) \to \mathcal{P}(\mathbb{N}^s \times \mathbb{N}_\infty)$, defined as follows:*

$$F(X) = \{(\bar{x}, 0) \mid \psi_b(\bar{x})\} \cup \{(\bar{x}, n+1) \mid \psi_r(\bar{x}, \bar{x}') \wedge (\bar{x}', n) \in X \text{ for some } \bar{x}' \in \mathbb{N}^s, n \in \mathbb{N}_\infty\}$$

We write $T_h = gfp\ F$.

As a notational convention, we consider relations $T_h \subseteq \mathbb{N}^s \times \mathbb{N}_\infty$ to be *multivalued functions* $T_h : \mathbb{N}^s \to \mathcal{P}(\mathbb{N}_\infty)$. Their *domain*, denoted by dom T_h, is the set of $\bar{x} \in \mathbb{N}^s$ such that $T_h(\bar{x}) \neq \emptyset$.

Example 2. The following CRS

$$\begin{array}{ll} T_h(x) = 0 & \{x = 0\} \\ T_h(x) = 1 + T_h(x') & \{x > 0 \wedge x' < x\} \end{array}$$

defines the relation $T_h = [0 \mapsto \{0\}, 1 \mapsto \{1\}, 2 \mapsto \{1, 2\}, \ldots, i \mapsto \{1..i\}, \ldots]$.

The choice of the greatest fixed point in Definition 2 is motivated by the need of capturing non-terminating call chains, as shown in the following example.

Example 3. Consider the following CRS:

$$\begin{array}{ll} T_h(x) = 0 & \{x = 0\} \\ T_h(x) = 1 + T_h(x') & \{x' > x\} \end{array}$$

Let us prove that it defines the following relation:

$$T_h = \{(0, 0), (0, +\infty), (1, +\infty), (2, +\infty), \ldots\}$$

The operator F, applied to this particular case, is defined as follows:

$$F(X) = \{(0, 0)\} \cup \{(x, n+1) \mid x < x' \wedge (x', n) \in X \text{ for some } x' \in \mathbb{N}, n \in \mathbb{N}_\infty\}$$

It is easy to see that $F(T_h) = T_h$. Hence T_h is a fixed point. Now we prove that is the greatest one by contradiction: assume that there exists a $T'_h \supset T_h$ such that $T'_h = F(T'_h)$. Since T'_h strictly extends T_h, we have two possibilities:

- $(x, 0) \in T'_h$ for some $x \neq 0$. This cannot happen, since $T'_h = F(T'_h)$ and the only tuple that F can return with a 0 in its right-hand side is $(0, 0)$.
- $(x, n) \in T'_h$ for some $x \geq 0$ and some n different from 0 and $+\infty$. Then, there must exist some $x_1 > x$ such that $(x_1, n-1) \in T'_h$, which leads to a contradiction if $n = 1$, as we have seen in the previous point. If $n > 1$, and since $(x_1, n-1) \in T'_h$, there exists another $x_2 > x_1$ such that $(x_2, n-2) \in T'_h$ and we apply the same reasoning as before. Eventually we will reach a tuple $(x_n, 0) \in T'_h$ for some $x_n > 0$, leading to a contradiction.

Therefore, the set T_h shown above is the relation defined by this CRS. Notice that the least fixed point of the F operator (which is $\{(0, 0)\}$) does not account for the sets of input values of T_h that lead to a non-terminating evaluation.

In order to apply the techniques explained in Sec. 3, it is necessary to choose a set of points and determine the maximum value returned by T_h when applied to each of these points. However, in general, it may be difficult to compute max $T_h(\bar{x})$ for an arbitrary $\bar{x}$, mainly due to non-determinism of CRSs. There may be a possibly infinite amount of vectors $\bar{x}' \in \mathbb{N}^s$ satisfying the recursive guard $\psi_r(\bar{x}, \bar{x}')$, and hence being eligible to be passed as argument to the recursive call to T_h.

Example 4. Assume the following CRS:

$$\begin{array}{ll} T_h(x) = 0 & \{x \geq 100\} \\ T_h(x) = 1 + T(x') & \{0 \leq x < 100, x < x'\} \end{array}$$

We get $T_h(0) = \{1..100\}$, but there are infinitely many ways of deriving $(0, 1) \in T_h$. In general, for any $x' \geq 100$ we obtain $(x', 0) \in T_h$ and hence $(0, 1) \in T_h$.

Given these difficulties, we will consider the evaluation of T_h in a bottom-up fashion: we start from the set of points A_0 such that the evaluation of T_h returns $\{0\}$. These points are known because they satisfy the base guard, but not the recursive one. In the next step, we consider the set of points that satisfy the recursive guard but, in the latter case, the corresponding recursive call falls into a base case. We denote by A_1 the set of these points together with those of A_0. In general, our aim is to find a hierarchy of sets $A_0 \subseteq A_1 \subseteq \ldots \subseteq A_i$, where each A_i contains the values of x such that the evaluation of $T_h(x)$ does not require more than i unfoldings. Recall that T_h can be viewed as an evaluation step counter for the original recurrence defining T.

Definition 3. *Given a relation $T_h : \mathbb{N}^s \to \mathcal{P}(\mathbb{N})$ and $i \in \mathbb{N}$, we define the set A_i as follows:*

$$A_i = \{\bar{x} \in \text{dom } T_h \mid \max T_h(\bar{x}) \leq i\}$$

Example 5. Back to our Example 2, we obtain $A_i = \{0..i\}$ for each $i \in \mathbb{N}$, whereas in Example 3 we get $A_i = \{0\}$ for each $i \in \mathbb{N}$.

Our next step is to find a characterization of these A_i sets in terms of the guards occurring in the CRS. This characterization is given as a set of predicates φ_i, defined as follows:

Definition 4. *Given the CRS in (3) and for each $i \in \mathbb{N}$, we define the predicate φ_i as follows:*

$$\varphi_0(\bar{x}) \stackrel{\text{def}}{=} \psi_b(\bar{x}) \wedge \forall \bar{x}'.\neg\psi_r(\bar{x}, \bar{x}')$$

$$\varphi_i(\bar{x}) \stackrel{\text{def}}{=} \varphi_0(\bar{x}) \vee [(\exists \bar{x}'.\psi_r(\bar{x}, \bar{x}')) \wedge \forall \bar{x}'.(\psi_r(\bar{x}, \bar{x}') \Rightarrow \varphi_{i-1}(\bar{x}'))] \qquad \textit{where } i > 0$$

If the constraints occurring in the guards of the CRS are linear, there exist efficient methods [10, 28] for removing the inner quantifiers of the φ_i predicates. In [19] a survey of these techniques can be found. The guards currently generated by COSTA are linear. Our implementation relies on linear programming applied to finite unions of polyhedra, which are handled by the Parma Polyhedra

Library (PPL) [6]. This technique is targeted towards real numbers (whereas our predicates are constrains on natural numbers), but it can be applied in our approach if our search of test nodes is restricted to points with natural coordinates. As an alternative, we can use a computer algebra system and its extension to a computer logic system [13, 11], which would also work in the case in which the guards are defined by nonlinear constraints.

Example 6. We get the following predicates from the CRS given in Example 2:

$$\begin{aligned}
\varphi_0(x) &\equiv x = 0 \wedge \neg\exists x'.(x > 0 \wedge x' < x) \\
&\equiv x = 0 \wedge \neg(x > 0) && \{\text{quantifier elimination}\} \\
&\equiv x = 0
\end{aligned}$$

$$\begin{aligned}
\varphi_1(x) &\equiv x = 0 \ \vee (x > 0 \wedge \forall x'.[x > 0 \wedge x' < x \Rightarrow x' = 0]) \\
&\equiv x = 0 \ \vee (x > 0 \wedge \neg[x > 0 \wedge 1 < x \wedge 1 \neq 0]) && \{\text{quantifier elimination}\} \\
&\equiv x = 0 \vee x = 1
\end{aligned}$$

$$\begin{aligned}
\varphi_2(x) &\equiv \ldots \\
&\equiv x \geq 0 \wedge x \leq 2
\end{aligned}$$

Now we prove that these predicates characterize the A_i sets. Without imposing special conditions on the CRSs, we can only prove that the φ_i predicates offer *sufficient* conditions for belonging to the A_i sets. More formally, $\{\bar{x} \in \mathbb{N}^s \mid \varphi_i(\bar{x})\} \subseteq A_i$ for each $i \in \mathbb{N}$. Strict inclusion may hold, in particular, when there are elements in the domain such that the evaluation of T_h gets stuck, as the following example shows.

Example 7. Given the following CRS:

$$\begin{array}{ll}
T_h(x) = 0 & \{x = 0\} \\
T_h(x) = 1 + T_h(x') & \{x \geq 2 \wedge (x' = 0 \vee x' = 1)\}
\end{array}$$

We get $T_h = [0 \mapsto \{0\}, 1 \mapsto \emptyset] \cup [i \mapsto \{1\} \mid i \geq 2]$ and hence $A_0 = \{0\}$ and $A_i = \{0, 2, 3, 4, \ldots\}$ for each $i \geq 1$. However, by applying the corresponding definition, we obtain $\varphi_i \equiv x = 0$ for each $i \in \mathbb{N}$, since the implication $\psi_r(x, x') \Rightarrow \varphi_{i-1}(x')$ does not hold when $x' = 1$. Our φ_i exactly approximates A_i only when $i = 0$.

We can ensure that the φ predicates actually characterise the A_i by imposing some mild conditions on our CRSs, namely, that every vector $\bar{x}$ satisfies at least one of the guards in the CRS.

Theorem 1. *Given the CRS in (3), assume that $\psi_b(\bar{x}) \vee \exists \bar{x}'.\psi_r(\bar{x}, \bar{x}')$ holds for every $\bar{x} \in \mathbb{N}^s$. If $T_h : \mathbb{N}^s \to \mathcal{P}(\mathbb{N}_\infty)$ is the relation defined by this CRS, the following holds for each $i \in \mathbb{N}$:*

$$A_i = \{\bar{x} \in \mathbb{N}^s \mid \varphi_i(\bar{x})\}$$

Proof. By induction on i. The condition $\psi_b(\bar{x}) \vee \exists \bar{x}'.\psi_r(\bar{x}, \bar{x}')$ is only needed for proving the $\subseteq$ inclusion. The $\supseteq$ inclusion holds without special provisions. □

5 Searching for Test Nodes: Gradient-based Method

Recall that $h(x) = \max T_h(\bar{x})$ by the definition, and s is the dimension of the vector $\bar{x} = (x_1, \ldots x_s)$. In this section we show how to construct a polynomial function $T_h^+(\bar{x}) \geq h(\bar{x})$.

The graph of $h(\bar{x})$ in $s+1$-dimensional space is presented via the collection of sets $A_i' := (A_i, i) = \{(\bar{x}, i) \mid \bar{x} \in A_i\}$, which, informally speaking, form *upside-down terraces*. This form is explained by the fact that $A_i \subseteq A_{i+1}$. An upper bound $T_h^+(\bar{x})$ "covers" the graph of h. Intuitively, the monotonicity behavior of a good upper bound and the monotonicity behavior of h coincide, in the sense that the gradient of the bound at a point on the edge A_i is the almost the same as the "gradient" of h at this point. The *gradient* of a smooth scalar function $f(\bar{x})$ at a point $\bar{x}$ shows *the direction of the greatest rate of increase* of the function. It is defined as the vector of the derivatives: $\nabla(f) := \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_s}\right)$. The graph of h is not smooth, therefore here the notion of the gradient at the point $\bar{x} \in A_i$ is intuitive and taken as *direction to the closest point on the next-level terrace*. This closest point belongs to $B_{i+1} := A_{i+1} \setminus A_i$. The gradient-based method of finding test nodes mimics climbing up from B_i to B_{i+1}: from a point $\bar{x}_i$ on B_i to $\bar{x}_{i+1}$ which is the *closest* to $\bar{x}_i$ point on B_{i+1}. Based on this intuition, we developed the procedure of finding test nodes. We have implemented it in Prolog, in order to incorporate it later into the COSTA implementation.

As a starting point we note that there is a (Prolog) procedure, that given guards, generates sets A_i, for arbitrary i, using their definition via φ_i. Sets A_i are given in a disjunctive normal form, that is as disjunctions of conjunctions of (in)equations of the form $F(\bar{x})\ b\ G(\bar{x})$, where b is one of $\{\geq, >, =, <, \leq\}$. In general, we do not demand that these atomic predicates are linear, if a used tool or library allows to deal with non-linear problems. In our current implementation, allowed atomic predicates are linear since we apply Parma Polyhedra Library, already used by COSTA team.

We need a set operation $(-)^*$ which is defined as follows. X^* is obtained from a disjunctive X by replacing all inequations of the form $F > G$ ($F < G$) with inequations $F \geq G + 1$ ($F \leq G - 1$).

- *Inputs:*
  - the degree d of a polynomial upper bound.
  - the closed sets A_i, $0 \leq i \leq l$ for some large enough l. (If some A_i is not closed, re-assign $A_i := (A_i)^*$.)
- *Output:* the set of interpolation nodes.

- *Procedure:*
  1. Construct $B_i^* := (A_i \setminus A_{i-1})^*$ for all $i > 1$, and assume B_0^* to be the boundary of A_0.
  2. Choose an initial level i_0, and let i be the level counter. The initial assignment is $i = i_0 + 1$.
  3. Pick up a point on $B_{i_0}^*$.

4. The next point $\bar{x}_i$ is computed as the closest to $\bar{x}_{i-1}$ point on B_i^*. Formally, $\bar{x}_i := argmin_{\bar{y}}\rho(\bar{x}_{i-1}, \bar{y})$, where $\bar{y} \in B_i^*$. In general, function ρ may be any topological distance, like Euclidean one. In our implementation, due to the fact that we are limited by linear PPL, ρ is a Manhattan distance.
5. Repeat the procedure for all $i = i_0 + 1, \ldots, l$.
6. If in all the constructed paths, including the current one, there are enough nodes satisfying s-dimensional NCA condition (e.g. on a grid), then stop. Otherwise go to step 2 and construct yet another path.

Now, let us see how this procedure works for the recurrence for Example 1. Recall, that here we deal with the following recursive steps:

$$\begin{array}{l} T_h(x,y) = 1 + T_h(x',y') \;\; \{x > 0 \wedge y = 0 \wedge x' = x - 1 \wedge y' = x - 1\} \\ T_h(x,y) = 1 + T_h(x',y') \;\; \{x \geq 0 \wedge y > 0 \wedge x' = x \wedge y' = y - 1\} \end{array}$$

For the example we assume $d = 2$. Moreover, it is enough to let $l = 3$ and consider the sets $A_0, \ldots, A_3$, in their disjunctive normal form:

$$\begin{array}{l|l} A_0 \; is & x = 0 \wedge y = 0 \\ A_1 \; is & A_0 \vee x - 1 = 0 \wedge y = 0 \vee x = 0 \wedge y - 1 = 0 \\ A_2 \; is & A_1 \vee x - 1 = 0 \wedge y - 1 = 0 \vee x = 0 \wedge y - 2 = 0 \\ A_3 \; is & A_2 \vee x - 1 = 0 \wedge y - 2 = 0 \vee x - 2 = 0 \wedge y = 0 \vee x = 0 \wedge y - 3 = 0 \end{array}$$

The implemented procedure constructs the following four paths:

$$\begin{array}{rl} (0,0), (1,0), (1,1), (1,2) & \text{given the initial point } (0,0) \text{ on } B_0^*. \\ (0,1), (1,1) & \text{given the initial point } (0,1) \text{ on } B_1^*. \\ (0,2), (1,2) & \text{given the initial point } (0,2) \text{ on } B_2^*. \\ (2,0) & \text{given the initial point } (2,0) \text{ on } B_3^*. \end{array}$$

These nodes coincide with the nodes that we have considered in the example in Sec. 3. They form an NCA configuration on a plane and define uniquely the interpolation polynomial $T_h^+(\bar{x}) = \dfrac{x^2}{2} + \dfrac{x}{2} + y$. It passes the checking shown in Sec. 6.

The obtained routes and the graph of $T_h^+(x,y)$ are given in Fig. 1.

The current implementation still needs to be optimised, first of all by simplifying the intermediate computations for $B_i = A_i \setminus A_{i-1}$, e.g. by applying set-theoretic axiomatics, such as $X \cup \emptyset = 0$ (removing inconsistent conjunctions) or $X \cup Y = X$ for $Y \subseteq X$. This will allow to shorten intermediate disjunctive forms. At the moment too long disjunctions cause stack overflow in the current implementation. Second, constraints A_i can be extended "on the fly" so that we avoid repeating points from already generated pats. Third, choice of initial nodes on paths is still manual, and we want to have it automatic.

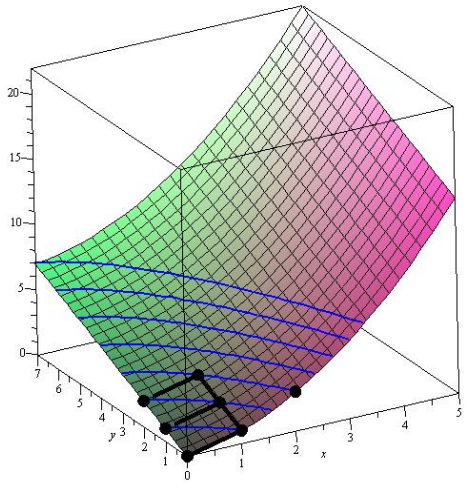

Fig. 1. The obtained routes and the graph of $T_h^+(\bar{x}) = \dfrac{x^2}{2} + \dfrac{x}{2} + y$ that in this case coincides with $h(x, y)$.

6 Proving the Bound Correct

In this section we describe the procedure that can be followed including the proof of the found bound is correct. First, we discuss what to do when the degree of the interpolated polynomial is different form the initial guess. Then, we describe methods to check whether the interpolated polynomial is indeed a correct bound. Finally, we describe how to proceed when such a check fails.

As it was pointed out in Sec. 1, the degree of the polynomial must be guessed in advance. Assume that we start the node search assuming a degree d. If the actual height of the call-tree $h(\bar{x})$ is described by a polynomial expression in $\bar{x}$ with that degree, then our interpolation-based approach returns exactly $h(\bar{x})$, provided the testing nodes are arranged in a **NCA** configuration. This is because $T_h^+(\bar{x})$ is an exact bound to $h(\bar{x})$ at the testing nodes, and the polynomial of degree d that interpolates these nodes is unique. Therefore, if T_h^+ and h coincide at the testing nodes, then $T_h^+(\bar{x}) = h(\bar{x})$ for all $\bar{x}$. If $h(\bar{x})$ is not given by a polynomial expression, there are several several possibilities:

- The previously guessed degree d may be lower than the actual degree of $h(\bar{x})$. In that case, the interpolation technique may result in a T_h^+ which is not an upper bound to h.
- In those cases in which $h(\bar{x})$ is not given by a polynomial expression, but it can be bounded by a polynomial, the heuristic-based choice of points shown in Sec. 5 increases the chances of obtaining a correct bound. This happens, in particular, when $h^+(\bar{x})$ is given by a step function, as in Fig. 2.

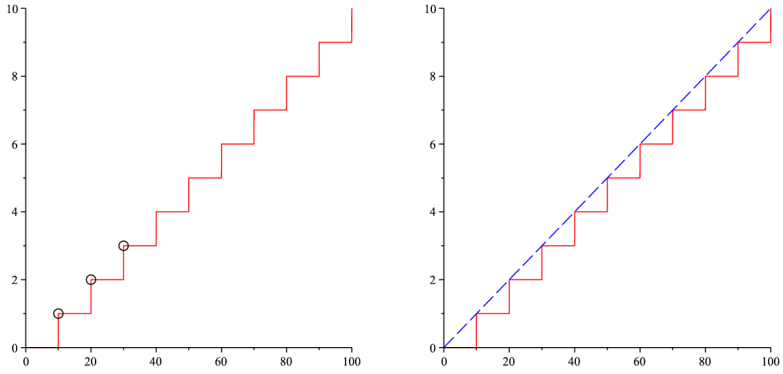

Fig. 2. If $h(\bar{x})$ is a step function, the choice of testing nodes at the left-hand side, results in the bound shown with a dashed line at the right-hand side, which is correct.

– Finally, it could be that $h^+(\bar{x})$ is not bounded by any polynomial, but by an expression of a higher complexity class (e.g. exponential). This case occurs very rarely in practice, and it is obvious that such upper bounds cannot be obtained by polynomial interpolation.

These scenarios above motivate the need to devise a way of checking whether the obtained T_h^+ is correct, so as to discard unsound bounds. A sufficient condition under which an upper bound $T_h^+(\bar{x})$ obtained by interpolation is an upper bound to the height $h(\bar{x})$ of the evaluation trees is given by the conjunction of:

$$\forall \bar{x} \,.\, T_h^+(\bar{x}) \geq 0$$
$$\forall \bar{x}, \bar{x}' \,.\, \psi_r(\bar{x}, \bar{x}') \Rightarrow (T_h^+(\bar{x}) \geq 1 + T_h^+(\bar{x}'))$$

Assuming that $\forall \bar{x} \,.\, h(\bar{x}) \neq \infty$ (otherwise, there would be no bound for $h(\bar{x})$), the proof is a straightforward induction on $h(\bar{x})$.

These predicates are decidable in Tarski's theory of real closed fields [26]. There are a number of tools available implementing improved versions of Tarski's procedure. For instance, QEPCAD [7] is free and offers an up-to-date version of Collins' algorithm [9]. If the predicates hold for real numbers, they will also hold for natural ones.

If checking fails, then one should either choose another set of test nodes, or increase the degree of a polynomial. If the degree d is too high at the beginning, one will need too many test points, and, therefore, the computation time will be longer. If the inferred polynomial has the lower degree than d, then the coefficients of higher-degree terms are equal to 0. A possible strategy is to start with a low degree such as $d = 2$, and then increase d at each iteration until either a degree succeeds or some time-out expires. In the latter case, we would report a fail to infer the bound.

7 Conclusions and Related Work

In this paper we have applied polynomial interpolation-based techniques in order to extend the PUBS recurrence solver, so that it can deal with a broader set of CRSs. This made it possible to propose an extension of the COSTA system replacing one of its static analyses, the ranking function analysis, by another kind of analysis, height analysis, in such a way that polynomial bounds of any degree may be inferred instead of just linear expressions.

Related Work

We have taken the work described in [2] as our point of reference. In a more recent work [4] the authors improve the precision of PUBS by considering worst- and best-case bounds to the cost of each loop iteration. The ideas described in this paper are orthogonal to those in [4] and can also be applied there.

In a different direction, COSTA has improved its memory analysis in order to take different models of garbage collection into account [3]. However, the authors claim that this extension does not require any changes to the recurrence solver PUBS. Thus, the techniques presented here should also fit with this extension.

In the field of functional languages, a seminal paper on static inference of memory bounds is [16]. A special type inference algorithm generates a set of linear constraints which, if satisfiable, they build a safe linear bound on the heap consumption.

One of the authors extended this type system in [14] in order to infer polynomial bounds. Surprisingly, the constraints resulting from the new type system are still linear ones. Although not every polynomial can be inferred by this system, the work was a remarkable step forward in the area. The language used is still functional, first-order and eager, but the resource inferred is a parameter. It could be either memory or time depending on some constants attached to the typing rules. A limitation of this work is that the inferred polynomials, even if they are multivariate ones, must not have multivariate terms. This limitation is removed in a more recent work [15].

The application of polynomial interpolation techniques makes it possible to derive polynomial complexity without any restriction in advance on the kind of polynomials. With interpolation polynomials can be multivariate and non-monotonic. For size analysis of functional languages several interpolation results have been developed in the AHA Project [12]. First, a size analysis type system is developed together with language constraints such that sized type checking can be shown to be decidable. With polynomial interpolation type inference is made possible [17]. The full sized type system is given in [21]. In [25] it is shown how the basic type system, which is defined for list structures only, can be extended to allow algebraic data types. The size analysis systems give precise size functions. It has been shown that also general polynomial lower and upper bounds can be derived using polynomial interpolation [22].

Polynomial interpolation has also been applied to non-functional languages. For Java an analysis was made to derive ranking functions for loops [24].

References

1. Albert, E., Arenas, P., Genaim, S., Puebla, G., Zanardini, D.: COSTA: Design and Implementation of a Cost and Termination Analyzer for Java Bytecode. In: Formal Methods for Components and Objects (FMCO'07). pp. 113–133. LNCS 5382, Springer (October 2008)
2. Albert, E., Arenas, P., Genaim, S., Puebla, G.: Closed-form upper bounds in static cost analysis. J. Autom. Reasoning 46(2), 161–203 (2011)
3. Albert, E., Genaim, S., Gómez-Zamalloa, M.: Parametric inference of memory requirements for garbage collected languages. In: Vitek, J., Lea, D. (eds.) ISMM. pp. 121–130. ACM (2010)
4. Albert, E., Genaim, S., Masud, A.N.: More precise yet widely applicable cost analysis. In: Jhala, R., Schmidt, D.A. (eds.) VMCAI. Lecture Notes in Computer Science, vol. 6538, pp. 38–53. Springer (2011)
5. Bagnara, R., Zaccagnini, A., Zolo, T.: The automatic solution of recurrence relations. I. Linear recurrences of finite order with constant coefficients. Quaderno 334, Dipartimento di Matematica, Università di Parma, Italy (2003), available at http://www.cs.unipr.it/Publications/
6. Bagnara, R., Ricci, E., Zaffanella, E., Hill, P.M.: Possibly not closed convex polyhedra and the Parma polyhedra library. In: Proceedings of the 9th International Symposium on Static Analysis, SAS 2002. pp. 213–229. Springer (2002)
7. Brown, C.W.: QEPCAD: Quantifier Elimination by Partial Cylindrical Algebraic Decomposition. http://www.cs.usna.edu/qepcad/B/QEPCAD.html (2004)
8. Chui, C.K., Lai, M.J.: Vandermonde determinants and lagrange interpolation in R^s. In: Nonlinear and Convex Analysis, Proceedings in Honor of Ky Fan. pp. 23–35. Marcel Dekker Inc., N.Y. (1987)
9. Collins, G.E.: Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In: Barkhage, H. (ed.) Automata Theory and Formal Languages. LNCS, vol. 33, pp. 134–183. Springer (1975)
10. Cooper, D.C.: Theorem proving in arithmetic without multiplication. Machine Inteligence 7, 91–100 (1972)
11. Dolzmann, A., Sturm, T.: Redlog user manual. Tech. Rep. MIP-9905, FMI, Universität Passau (1999), edition 2.0 for Version 2.0
12. van Eekelen, M., Shkaravska, O., van Kesteren, R., Jacobs, B., Poll, E., Smetsers, S.: AHA: Amortized Heap space usage Analysis. In: Morazán, M. (ed.) Selected revised papers of the 8th international symposium on Trends in Functional Programming (TFP'07). pp. 36–53. Intellect, New York, USA (2007)
13. Hearn, A.C.: REDUCE. User's Manual. Version 3.8 (2004)
14. Hoffmann, J., Hofmann, M.: Amortized Resource Analysis with Polynomial Potential. A Static Inference of Polynomial Bounds for Functional Programs. In: ESOP 2010, LNCS 6012. pp. 287–306. Springer (2010)
15. Hoffmann, J., Aehlig, K., Hofmann, M.: Multivariate amortized resource analysis. In: Ball, T., Sagiv, M. (eds.) POPL. pp. 357–370. ACM (2011)
16. Hofmann, M., Jost, S.: Static prediction of heap space usage for first-order functional programs. In: Proc. 30th ACM Symp. on Principles of Programming Languages, POPL'03. pp. 185–197. ACM Press (2003)
17. van Kesteren, R., Shkaravska, O., van Eekelen, M.: Inferring static nonmonotonically sized types through testing. In: Proceedings of 16th international workshop on Functional and (Constraint) Logic Programming (WFLP'07). Electronic Notes in Theoretical Computer Science, vol. 216C, pp. 45–63. Paris, France (2007)

18. Lucas, S.: Polynomials over the reals in proofs of termination: from theory to practice. RAIRO Theoretical Informatics and Applications 39(3), 547–586 (2005)
19. Nipkow, T.: Linear quantifier elimination. J. Autom. Reasoning 45(2), 189–212 (2010)
20. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: Steffen, B., Levi, G. (eds.) VMCAI. Lecture Notes in Computer Science, vol. 2937, pp. 239–251. Springer (2004)
21. Shkaravska, O., van Eekelen, M., van Kesteren, R.: Polynomial size analysis of first-order shapely functions. Logical Methods in Computer Science 5(2:10), 1–35 (2009), selected Papers from TLCA 2007
22. Shkaravska, O., van Eekelen, M., Tamalet, A.: Collected size semantics for functional programs over lists. In: Scholz, S.B. (ed.) Revised selected papers of the 20th international symposium on Implementation and Application of Functional Programming (IFL'08). LNCS, vol. 5836. Springer-Verlag, University of Hertfordshire, UK (2011), to appear.
23. Shkaravska, O., van Eekelen, M.C.J.D., van Kesteren, R.: Polynomial size analysis of first-order shapely functions. Logical Methods in Computer Science 5(2) (2009)
24. Shkaravska, O., Kersten, R., van Eekelen, M.: Test-based inference of polynomial loop-bound functions. In: Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java, PPPJ 2010. ACM (2010)
25. Tamalet, A., Shkaravska, O., van Eekelen, M.: Size analysis of algebraic data types. In: Achten, P., Koopman, P., Morazán, M.T. (eds.) Selected revised papers of the 9th international symposium on Trends in Functional Programming (TFP'08). pp. 33–48. Intellect (2009)
26. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. University of California Press, Berkeley (1948)
27. Wegbreit, B.: Mechanical program analysis. Commun. ACM 18(9), 528–539 (1975)
28. Weispfenning, V.: The complexity of linear problems in fields. J. Symb. Comput. 5(1/2), 3–27 (1988)