Symbolic Polynomial Maximization Over Convex Sets and Its Application to Memory Requirement Estimation

Philippe Clauss, Federico Javier Fernández, Diego Garbervetsky, and Sven Verdoolaege

***Abstract*—Memory requirement estimation is an important issue in the development of embedded systems, since memory directly influences performance, cost and power consumption. It is therefore crucial to have tools that automatically compute accurate estimates of the memory requirements of programs to better control the development process and avoid some catastrophic execution exceptions. Many important memory issues can be expressed as the problem of maximizing a parametric polynomial defined over a parametric convex domain. Bernstein expansion is a technique that has been used to compute upper bounds on polynomials defined over intervals and parametric "boxes". In this paper, we propose an extension of this theory to more general parametric convex domains and illustrate its applicability to the resolution of memory issues with several application examples.**

***Index Terms*—Bernstein expansion, convex polytopes, memory requirement, program optimization, static program analysis.**

I. Introduction

THE determination of the amount of memory required by a program through static analysis has received a lot of attention in recent years [1]–[6]. Usually, the first step is to determine the amount of memory "in use" at a given point during the execution of the program. The memory requirement is then obtained by computing the maximum of the resulting expression over all such points.

In particular, if the program consists of a sequence of loop nests with loop bounds and array references that are affine functions of the enclosing loop iterators and structural parameters, then the iterations of the loops can be represented by the integer points in parametric polytopes. This representation is known as the polytope model [7]. The memory in use at a given loop iteration can usually be represented or approximated by a polynomial in both the loop iterators and the structural parameters. The problem of calculating the memory requirements of a program then reduces to computing the maximum of a polynomial over all integer points in a parametric polytope, resulting in an expression that only depends on the structural parameters. This maximization of a polynomial over a parametric polytope also has applications in extending static analysis beyond the polytope model [8].

De Loera *et al.* [9] have recently shown that maximizing an arbitrary polynomial over the integer points in a non-parametric polytope is NP-hard, while also giving a fully polynomial-time approximation scheme for computing this maximum when the polynomial is non-negative and the dimension of the polytope is fixed. However, to the best of our knowledge, their algorithm has not been implemented yet and it cannot easily be extended to the parametric case. This evidence suggests that the exact parametric maximum over the integer points in a parametric polytope may not in general be easily computable. We therefore relax our problem first by computing the maximum over all *rational* points instead of all integer points and second by computing an *upper bound* rather than the maximum. In particular, we will use an extension of *Bernstein expansion* to parametric polytopes to compute these upper bounds. The resulting upper bounds will usually be fairly accurate and we can *detect* whether we have computed the actual maximum or not.

Classical Bernstein expansion [10], [11] allows for the determination of bounds on the range of a multivariate polynomial considered over a box [8], [12], [13]. Numerical applications of this theory have been proposed to the resolution of systems of strict polynomial inequalities [14], [15]. A symbolic approach to Bernstein expansion used in program analysis has also been proposed in [8], but only for "parametric boxes". This approach is subsumed by our approach. It has been shown that Bernstein expansion is generally more accurate than classic interval methods [16].

In this paper, we propose an extension of the theory of Bernstein expansion to handle multivariate parametric polynomials defined over parametric convex polytopes. These parametric polytopes can be described as the convex hull of a finite set of parametric generators, as the solution set of a finite number of linear constraints over the variables and the parameters or as a parametric box. Then we use this extension to compute upper bounds for multivariate polynomials modeling the memory usage of programs. More precisely, it is shown how the described technique can be used to compute bounds on the memory consumption of programs.

II. Illustrative Example

As a typical example of the kind of memory requirement estimation problems that can be handled using our technique, we

Manuscript received December 05, 2007; revised March 23, 2008. First published May 02, 2009; current version published July 22, 2009.

P. Clauss is with the Laboratory ICPS-LSIIT, Université Louis Pasteur, 67400 Illkirch, France (e-mail: clauss@icps.u-strasbg.fr).

F. J. Fernández and D. Garbervetsky are with the Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina.

S. Verdoolaege is with the Leiden Institute of Advanced Computer Science, Universiteit Leiden, 2333 CA Leiden, The Netherlands, and also with the Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium.

Digital Object Identifier 10.1109/TVLSI.2008.2002049

```
1 for (i = 0; i < 4*n-1; ++i)
      for (j = 0; j < n; ++j) {
          if (i+j >= n-1 && i+j <= 3*n-2)
              a[i][j] = f(i,j);
          if (i+j >= 2*n-1 && i+j <= 4*n-2)
6             b[j] = b[j] + a[i-n][j];
      }
```

Fig. 1. A nested loop with temporary array a.

consider the problem of finding the maximal number of live elements during the course of a program, where an element is "live" at a given point in the program if it has been defined (written) and still needs to be used (read). A bound on the maximal number of live elements is an indication of the amount of memory required for the execution of the program.

Assume that array a in the (slightly contrived) code fragment of Fig. 1 is a temporary array that is only used inside the given loop nest. Let us concentrate on the subproblem of finding the maximal number of live elements in the array a. In this simple example, each array element that is defined in line 4 is used exactly once in line 6 and is therefore live between its definition and its first and only use. The number of live elements therefore reaches a maximum right before some element is used in the second statement. The general case of finding the maximal number of live elements, i.e., when there is no one-to-one mapping between the iteration domains and the array space or when there are several non-disjoint array references consuming elements from the same array, is explained in Section VIII-A.

Fig. 2 shows the iteration domains of the two statements and the elements of array a live right before iteration $(n+3, n-2)$ of the second statement. Each array element is shown at the iteration defining it. For any given iteration (i,j) in the iteration domain $D_2(n)$ of the second statement, i.e., the set of iterations for which the statement is executed, the number of elements live before the given iteration is equal to the number of elements defined before that iteration (in this case, the number of iterations (i',j') in the iteration domain $D_1(n)$ of the first statement that precede iteration (i,j) of the second statement) minus the number of elements used before that iteration (i.e., the number of iterations $(i',j') \in D_2(n)$ that precede iteration (i,j)). Using standard techniques to extract affine constraints from code, available in several modern industrial and research compilers, e.g., [17]–[19], these two iteration domains can be represented as

$$D_1(n) = \left\{ (i,j) \mid \begin{array}{l} 0 \le i < 4n-1 \\ 0 \le j < n \\ n-1 \le i+j \le 3n-2 \end{array} \right\}$$

and

$$D_2(n) = \left\{ (i,j) \mid \begin{array}{l} 0 \le i < 4n-1 \\ 0 \le j < n \\ 2n-1 \le i+j \le 4n-2 \end{array} \right\},$$

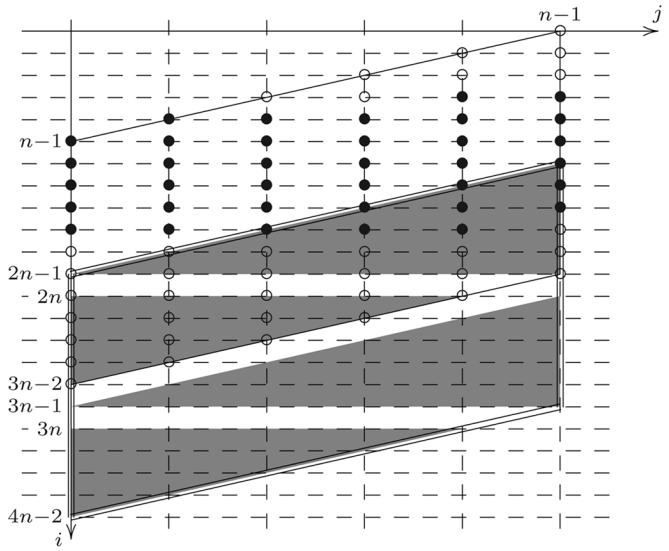

Fig. 2. Elements of array alive before iteration $(n+3, n-2)$ of the read.

and the number of elements live before a given iteration $(i,j) \in D_2(n)$ is then given by

$$\begin{aligned} L(n,i,j) = &\#\{(i',j') \in D_1(n) \mid (i',j') \preccurlyeq (i,j)\} \\ &- \#\{(i',j') \in D_2(n) \mid (i',j') \prec (i,j)\} \end{aligned} \quad (1)$$

where $\prec$ denotes the relation "lexicographically smaller than", i.e., "is executed before". The maximal number of live elements is then simply

$$M(n) = \max_{(i,j) \in D_2(n)} L(n,i,j).$$

Each of two sets in (1) can be described as a (disjoint) union of two sets bounded by affine constraints by expanding the lexicographical constraints as $(i',j') \prec (i,j) \equiv i' < i \vee (i' = i \wedge j' < j)$. Counting the number of elements in such sets, i.e., the number of integer points in parametric polytopes, can be performed very efficiently using existing techniques [20] and the result for $L(n,i,j)$ is the piecewise polynomial shown in (2) at the bottom of the page. The "pieces" of this piecewise polynomial are shown in grey in Fig. 2.

In order to compute an (over)estimate of the maximal number of live elements $M(n)$ we therefore need to compute an upper bound $U(n)$ on the piecewise polynomial (2). Using our extension of Bernstein expansion to parametric polytopes, we will be able to determine that

$$U(n) = n^2 + \frac{n}{4} + \frac{3}{4}, \qquad \text{if } n \ge 1 \quad (3)$$

is such an upper bound. Note that the actual maximum is $n^2 + 1$, so the relative error is small for large n. In Section X, we

$$\begin{cases} 2ni - n^2 + 3n - \frac{1}{2}i^2 - \frac{3}{2}i, & \text{if } (i,j) \in D_2(n) \wedge i \le 2n-1 \\ -\frac{1}{2}i^2 + 2ni - \frac{1}{2}i - n^2 + n + 1, & \text{if } (i,j) \in D_2(n) \wedge i \ge 2n \wedge i+j \le 3n-2 \\ -\frac{1}{2}i^2 + 2ni - \frac{3}{2}i - n^2 + 4n - j, & \text{if } (i,j) \in D_2(n) \wedge i+j \ge 3n-2 \wedge i \le 3n-1 \\ 8n^2 + \frac{1}{2}i^2 - 4ni + \frac{1}{2}i - j - 2n + 1, & \text{if } (i,j) \in D_2(n) \wedge i \ge 3n. \end{cases} \quad (2)$$

will see that a technique based on range propagation produces significantly less accurate results.

The rest of this paper is organized as follows. An overview of the general use of Bernstein expansion to compute an upper bound on the number of elements in a set described by linear constraints is given in Section III, with details on the extension of classical Bernstein expansion over an interval to expansion over convex polytope detailed in Section IV and a further computation step to simplify the resulting expressions in Section V. In Section VI, we show how to apply Bernstein expansion incrementally and in Section VII, we briefly give some additional information about our software implementation. Section VIII is devoted to the description of several interesting applications of the Bernstein approach to the analysis of memory behavior. Finally, we compare with related work in Section IX, show some experiments in Section X, and conclude in Section XI.

III. General Problem Formulation

The method used in Section II to compute (an upper bound on) the maximal number of live elements in a program, was to first compute the number of live elements at any given point in the program and then to compute an upper bound of the resulting expression over all program points. We can apply the same strategy to many memory requirement estimation problems. That is, we first compute the number of elements that satisfy some conditions and then compute an upper bound of the resulting expression.

A. An Upper Bound on the Number of Elements in a Set

In particular, we will be interested in the case where the elements are integer vectors and the conditions can be described by *linear* constraints. The first step is then to compute the number of elements $f(\mathbf{p},\mathbf{q})$ of some set $S(\mathbf{x},\mathbf{p},\mathbf{q})$, i.e.,

$$f(\mathbf{p},\mathbf{q}) = \#\{\mathbf{x}\in\mathbb{Z}^n \mid \exists \mathbf{y}\in\mathbb{Z}^{n'} : p(\mathbf{x},\mathbf{y},\mathbf{p},\mathbf{q})\} \quad (4)$$

where $p(\mathbf{x},\mathbf{y},\mathbf{p},\mathbf{q})$ is a conjunction of m linear constraints on $\mathbf{x}$, $\mathbf{y}$, $\mathbf{p}$ and $\mathbf{q}$,

$$p(\mathbf{x},\mathbf{y},\mathbf{p},\mathbf{q}) \iff A\mathbf{x}+B\mathbf{y}\geq C\mathbf{p}+D\mathbf{q}+\mathbf{f}$$

with $A\in\mathbb{Z}^{m\times n}$, $B\in\mathbb{Z}^{m\times n'}$, $C\in\mathbb{Z}^{m\times r}$, $D\in\mathbb{Z}^{m\times r'}$ and $\mathbf{f}\in\mathbb{Z}^m$. In the second step, we compute an upper bound on $f(\mathbf{p},\mathbf{q})$. That is, we compute a $U(\mathbf{q})$ such that

$$U(\mathbf{q}) \geq M(\mathbf{q}) = \max_{\mathbf{p}\in Q(\mathbf{q})} f(\mathbf{p},\mathbf{q}) \quad \text{for all } \mathbf{q} \quad (5)$$

with Q the domain of f and where we use the shorthand $Q(\mathbf{q}) = \{\,\mathbf{p} \mid (\mathbf{p},\mathbf{q})\in Q\,\}$. The first problem (4) is a counting problem while the second problem (5) is a "maximization" problem. Both of these problems are *parametric*, i.e., the result is not simply a number, but rather an expression in a number of parameters. The variables that act as parameters in one problem are, however, not the same as those that act as parameters in the other problem. In general, we can identify four sets of variables in the two problems:

1) the variables that are existentially quantified in the counting problem (4); in the example of Section II, there are no such variables;
2) the elements that need to be counted; in the example, these are the indices of the array or the iteration in which they are defined or used;
3) the variables over which the maximum needs to be taken; in the example these are the iterations of $D_2(n)$;
4) the structural parameters; in the example, there is a single structural parameter $\mathtt{n}$.

The latter two sets of variables will be parameters for the counting problem, while only the structural parameters will be parameters for the maximization problem. If there are no existentially quantified variables $\mathbf{y}$, i.e., if $n'=0$, then the set in (4) is called a *parametric polyhedron*. Actually, the set corresponds to the integer points in a polyhedron, the polyhedron itself being defined over the rationals or the reals, but in the remainder of the paper it will be implied that we are dealing with the integer points in such sets. If the polyhedron is bounded for each value of the parameters (which will usually be the case when we want to count the number of integer points in the polyhedron), then the set is called a *parametric polytope*. More specifically, we call these sets *$\mathcal{H}$-parametric polytopes*, since the constraints (which correspond to the "hyperplanes" that bound the polytope) are parametric in this description. Counting the number of integer points in such $\mathcal{H}$-parametric polytopes can be performed very efficiently using Barvinok's algorithm [20]. Otherwise, i.e., if $n'\neq 0$, then additional techniques, e.g., [21], [22], need to be applied, which still work fairly well in practice.

In either case, the result of this counting problem is a piecewise quasi-polynomial

$$f(\mathbf{p},\mathbf{q}) = \begin{cases} f_1(\mathbf{p},\mathbf{q}) & \text{if } (\mathbf{p},\mathbf{q})\in Q_1 \\ \dots & \\ f_M(\mathbf{p},\mathbf{q}) & \text{if } (\mathbf{p},\mathbf{q})\in Q_M \end{cases} \quad (6)$$

i.e., a subdivision of the parameter space Q (of the counting problem), with a quasi-polynomial $f_i(\mathbf{p},\mathbf{q})$ associated to each cell Q_i of the subdivision. These piecewise quasi-polynomials that result from counting problems are also called Ehrhart polynomial by some authors, e.g., [23]. The cells Q_i in the subdivision are themselves polyhedra, while a quasi-polynomial is a polynomial expression where the coefficients depend periodically on the variables. We can, however, avoid this periodicity (i.e., obtain an actual polynomial) by approximating the original parametric polytope to obtain either an underestimate or an overestimate [24]. In the example of Section II, four counting problems of the type (4) have to be solved, two for each of the sets in (1). Adding and subtracting the results, we obtain the expression in (2). Note that we do not need to deal with periodicity in this example, i.e., we obtain a piecewise *polynomial*.

To compute $U(\mathbf{q})$ in (5), we first compute

$$U_i(\mathbf{q}) \geq M_i(\mathbf{q}) = \max_{\mathbf{p}\in Q_i(\mathbf{q})} f_i(\mathbf{p},\mathbf{q}) \quad \text{for all } \mathbf{q}.$$

Note that Q_i is interpreted here as a parametric polyhedron with only the $\mathbf{q}$ as parameters. As in the counting problem, we may assume that Q_i is a parametric *polytope*. Finally $U(\mathbf{q})$ is constructed such that

$$U(\mathbf{q}) \geq U_i(\mathbf{q}) \quad \text{for all } i \text{ and for all } \mathbf{q}. \quad (7)$$

Both of these computations are discussed next.

B. An Upper Bound of a Polynomial Over a Parametric Polytope

Given a polynomial $f(\mathbf{p},\mathbf{q})$ defined over a parametric polytope $Q(\mathbf{q})$, the procedure described in this section will compute a subdivision of the parameter domain of $Q(\mathbf{q})$ into a finite set of cells R_j and for each R_j a *finite* set of polynomials $g_{j,k}(\mathbf{q})$ such that for

$$U(\mathbf{q}) = \begin{cases} \max_k g_{1,k}(\mathbf{q}), & \text{if } \mathbf{q} \in R_1 \\ \dots & \\ \max_k g_{N,k}(\mathbf{q}), & \text{if } \mathbf{q} \in R_N \end{cases} \tag{8}$$

we have

$$U(\mathbf{q}) \geq M(\mathbf{q}) = \max_{\mathbf{p}\in Q(\mathbf{q})} f(\mathbf{p},\mathbf{q}) \quad \text{for all } \mathbf{q}.$$

The procedure consists of the following steps, explained in more detail below.

- Obtain a representation of the parametric polytope as a convex combination of *parametric vertices*. This representation may be different over different cells R_j of the parameter domain.
- For each cell R_j, compute the *Bernstein coefficients* of the polynomial $f(\mathbf{p},\mathbf{q})$ corresponding to the parametric vertices describing the polytope over this cell. These Bernstein coefficients will form a superset $g'_{j,l}$ of the final $g_{j,k}$.
- Remove those polynomials $g'_{j,l}$ that are redundant with respect to other polynomials.

As we will see in Section IV, our algorithm for computing Bernstein coefficients expects the domain to be specified as a $\mathcal{V}$-parametric polytope $P : D \to \mathbb{Q}^r : \mathbf{q} \mapsto P(\mathbf{q})$,

$$P(\mathbf{q}) = \left\{ \mathbf{p} \mid \exists \alpha_i \in \mathbb{Q}_{\geq 0} : \mathbf{p} = \sum_i \alpha_i \mathbf{v}_i(\mathbf{q}), \sum_i \alpha_i = 1 \right\} \tag{9}$$

i.e., as a convex combination of some parametric generators $\mathbf{v}_i(\mathbf{q})$ for each $\mathbf{q}$ in the parameter domain $D \subset \mathbb{Q}^{r'}$. If for some value of $\mathbf{q}$, we have that $\mathbf{v}_i(\mathbf{q})$ is not a convex combination of the other $\mathbf{v}_j(\mathbf{q})$, then $\mathbf{v}_i(\mathbf{q})$ is called a *parametric vertex* of the polytope. In Section III-A, however, each of the Q_i is specified as a $\mathcal{H}$-parametric polytope instead, i.e.,

$$Q : D' \to \mathbb{Q}^r : \mathbf{q} \mapsto Q(\mathbf{q}) = \{\mathbf{p} \in \mathbb{Q}^r \mid A\mathbf{p} \geq B\mathbf{q} + \mathbf{d}\} \tag{10}$$

with $D' \subset \mathbb{Q}^{r'}$, $A \in \mathbb{Z}^{m\times r}$, $B \in \mathbb{Z}^{m\times r'}$ and $\mathbf{d} \in \mathbb{Z}^m$. Fortunately, since the constraints in (10) are linear in both the variables $\mathbf{p}$ and the parameters $\mathbf{q}$, the required vertices can be computed using PolyLib [25]. The result is a subdivision of the parameter space in polyhedral cells R_j, called *chambers*, each with an associated set of parametric vertices as in (9), such that for all $\mathbf{q} \in R_j$ we have $Q(\mathbf{q}) = P(\mathbf{q})$ [23]. Note that if the constraints describing the domain are only linear in the variables (and not in the parameters), then we may still compute the vertices of the domain, but the subdomains of the parameter space that have a fixed set of parametric vertices will no longer be polyhedral [26].

For example, the domain of the third case in (2) can be described as

$$D(n) = \left\{ (i,j) \mid \begin{array}{c} 0 \leq i \leq 3n-1 \\ 0 \leq j \leq n-1 \\ 3n-1 \leq i+j \leq 4n-2 \end{array} \right\} \tag{11}$$

where n is now the only parameter. In this case, there is only one parameter domain and we find the vertices

$$\left\{ \begin{pmatrix} 2n \\ n-1 \end{pmatrix}, \begin{pmatrix} 3n-1 \\ 0 \end{pmatrix}, \begin{pmatrix} 3n-1 \\ n-1 \end{pmatrix} \right\}, \text{ if } n \geq 1. \tag{12}$$

If there are multiple parameter domains, then the Bernstein coefficients are computed on each domain separately.

The computation of Bernstein coefficients will be explained in detail in Section IV. Here, we will only give an example of the result of such a computation. Suppose we want to compute an upper bound $U(n)$ for the polynomial

$$-\frac{1}{2}i^2 - \frac{3}{2}i - j - n^2 + 4n + 2in \tag{13}$$

over the domain with vertices (12), i.e., the third case of (2). The procedure of Section IV will produce as upper bound

$$U(n) = \max\left\{ n^2 - \frac{n}{4} + \frac{5}{4}, \frac{n^2}{2} + \frac{n}{2} + 1, \frac{n^2}{2} + \frac{3}{2}, n^2 + 1, \right. \\ \left. \frac{n^2}{2} - \frac{n}{2} + 2, n^2 - \frac{3n}{4} + \frac{7}{4} \right\}, \quad \text{if } n \geq 1. \tag{14}$$

To compute the upper bound for any particular value of n, we therefore need to evaluate these 6 polynomials at this value and take the maximum. However, it is clear that some of these polynomials are redundant in the sense that for any value of the parameters in the domain the polynomial always evaluates to a smaller number than some other polynomial. The removal of these redundant polynomials will be the subject of Section V. In our example, we will be able to simplify (14) to

$$U(n) = n^2 + 1, \quad \text{if } n \geq 1. \tag{15}$$

C. Combining Upper Bounds

In Section III-A, we explained how to compute an upper bound on a set of the form (4), by first solving a parametric counting problem, resulting in a piecewise (quasi-)polynomial, and then computing an upper bound on this piecewise polynomial. Section III-B showed how to compute an upper bound of a single polynomial over a single convex domain. Here, we explain how to combine the results to obtain the global upper bound (7).

For each polynomial $f_i(\mathbf{p},\mathbf{q})$ defined over the parametric polytope $Q_i(\mathbf{q})$ in (6), we compute the upper bound $U_i(\mathbf{q})$ (8). Note that the list of polynomials $g_{i,j,k}(\mathbf{q})$ may be the full list of potential upper bounds including the redundant ones since we are going to remove the redundant upper bounds in a later step. For each i, the $R_{i,j}$ form a polyhedral subdivision of the definition domain of $U_i(\mathbf{q})$. For the combined solution, we compute

Algorithm 1: Upper bound of a piecewise polynomial

Input: piecewise polynomial $f(\mathbf{p},\mathbf{q})$ (6)
Output: piecewise maximum of polynomials $U(\mathbf{q})$ (8) satisfying (5)

foreach $i \in [1,M]$ **do**
  $\left\{\left(R'_{i,j}, V_{i,j}(\mathbf{q})\right)\right\}_{j=1}^{N_i} \leftarrow$ ParametricVertices(Q_i) [25]
  foreach $j \in [1,N_i]$ **do**
    $g'_{i,j} \leftarrow$ BernsteinCoefficients$(R'_{i,j}, V_{i,j}(\mathbf{q}))$ (Section IV)
  foreach $\mathbf{j} = (j_1,\ldots,j_N)$, $0 \le j_i \le N_i$ **do**
    $R_{\mathbf{j}} \leftarrow \bigcap_{i,j_i\neq 0} R'_{i,j} \setminus \left(\bigcup_{i,j_i=0} \bigcup_{j=1}^{N_i} R'_{i,j}\right)$
    $\{g_{\mathbf{j},k}\}_k \leftarrow$ RemoveRedundant $\left(\bigcup_{i,j_i\neq 0} g'_{i,j_i}\right)$ (Section V)
  return $U(\mathbf{q}) = \left\{_{\mathbf{j}} \max_k g_{\mathbf{j},k}(\mathbf{q}) \quad \text{if } \mathbf{q} \in R_{\mathbf{j}}\right.$

the *common refinement* of these polyhedral subdivisions, i.e., a polyhedral subdivision R'_l of $\bigcup_i U_i$ such that for each cell R'_l and each i, either R'_l does not intersect with the domain of U_i or R'_l is contained in one of the $R_{i,j}$.

Having computed the common refinement, we associate to each R'_l the union of the sets of polynomials associated to each $R_{i,j}$ containing R'_l and obtain an expression for the global $U(\mathbf{q})$ of the form (8). Here, again, we can remove the polynomials in this union that are redundant with respect to the other polynomials. It is clear that if each $U_i(\mathbf{q})$ is an upper bound of $f_i(\mathbf{p},\mathbf{q})$ over $Q_i(\mathbf{q})$, then the function $U(\mathbf{q})$ constructed in this way is an upper bound for $f(\mathbf{p},\mathbf{q})$, satisfying (7).

Example 1: For the illustrative example (2) of Section II, we have already computed the upper bound $U_3(n)$ for the third case in (14). For the other three cases, we similarly obtain

$$U_1(n) = \max\left\{n^2+\frac{n}{4}+\frac{3}{4}, n^2+1, \frac{n^2}{2}+\frac{3}{2}n\right\}, \text{ if } n \ge 1$$

$$U_2(n) = \max\left\{n^2+1, n^2-\frac{n}{4}+\frac{3}{2}, \frac{n^2}{2}+\frac{3}{2}n\right\}, \text{ if } n \ge 2$$

$$U_4(n) = \max\left\{2, \frac{n^2}{2}-\frac{3}{2}n+3, \frac{n^2}{2}-n+2, \frac{3}{4}n+\frac{1}{2}, \frac{n^2}{2}-\frac{n}{2}+1, \frac{n}{4}+\frac{3}{2}\right\}, \text{ if } n \ge 2.$$

In this example, the common refinement consists of two cells: $n = 1$, with polynomials from $U_1(n)$ and $U_3(n)$ only, and $n \ge 2$, with polynomials from all $U_i(n)$. In the first cell, we can evaluate the polynomials and the upper bound is simply 2, while in the second cell we can remove redundant polynomials as described in Section V, and the only remaining polynomial is $n^2+n/4+3/4$. This upper bound on the number of live elements can be simplified to the expression shown in (3).

IV. Symbolic Bernstein Expansion Over a Convex Polytope

This section explains the theory behind Bernstein expansion. We first recall the classical Bernstein expansion of a univariate polynomial over an interval and then show how it can be extended to multivariate parametric polynomials over parametric convex polytopes.

A. Bernstein Expansion Over an Interval

There are many ways to represent a (rational) univariate degree-d polynomial $p(x) \in \mathbb{Q}[x]$. The canonical representation of $p(x)$ is as a $\mathbb{Q}$-linear combination of the power base, i.e., the powers of x

$$p(x) = \sum_{i=0}^{d} a_i x^i \tag{16}$$

with $a_i \in \mathbb{Q}$. The polynomial $p(x)$ can also be represented as a $\mathbb{Q}$-linear combination of the degree-d *Bernstein base polynomials* [10]–[13]

$$p(x) = \sum_{k=0}^{d} b_k^d B_k^d(x) \tag{17}$$

where the Bernstein polynomials $B_i^d(x)$ are defined by

$$B_k^d(x) = \binom{d}{k} x^k (1-x)^{d-k} \text{ with } \binom{d}{k} = \frac{d!}{k!(d-k)!} \tag{18}$$

for $k = 0,1,\ldots,d$, and $b_i^d \in \mathbb{Q}$ are the Bernstein coefficients corresponding to the degree-d basis.

Example 2: Here is an example of a univariate polynomial in its power form and in its Bernstein form:

$$\begin{aligned} p(x) &= x^3 - 5x^2 + 2x + 4 \\ &= 4B_0^3(x) + \frac{14}{3}B_1^3(x) + \frac{11}{3}B_2^3(x) + 2B_3^3(x) \end{aligned}$$

where $B_0^3(x) = (1-x)^3$, $B_1^3(x) = 3x(1-x)^2$, $B_2^3(x) = 3x^2(1-x)$ and $B_3^3(x) = x^3$. We will explain below how to compute the Bernstein coefficients in this expression.

The Bernstein expansion of a polynomial has many interesting properties. The properties that will interest us most here is that the sum of the Bernstein base polynomials (18) is 1 and that, on the interval $[0,1]$, $0 \le B_k^d(x) \le 1$. The first property follows from the identity

$$1 = (x+(1-x))^d = \sum_{k=0}^{d} B_k^d(x).$$

On the interval $[0,1]$, (17) expresses the polynomial $p(x)$ as a convex combination (with coefficients $B_i^d(x)$) of the Bernstein coefficients b_i^d. On this interval, the polynomial $p(x)$ is therefore bounded by its Bernstein coefficients, i.e.,

$$\min_{0\le i\le d} b_i^d \le p(x) \le \max_{0\le i\le d} b_i^d.$$

Moreover, if the minimum or maximum of the b_i^d is b_0^d or b_d^d then this bound is exact, since they correspond to values taken by $p(x)$ at the vertices as is clear from (18). These coefficients where the bound is exact are sometimes referred to as *sharp coefficients*.

Ratschek and Rokne [27] proved that the estimation error can be made smaller as the degree d is elevated. Hence, tighter

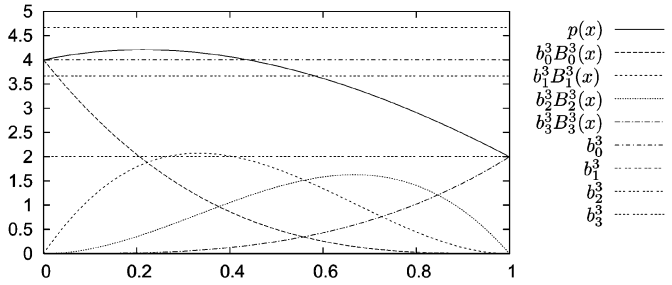

Fig. 3. Decomposition of the polynomial $p(x) = x^3 - 5x^2 + 2x + 4$ in the Bernstein basis.

bounds can be obtained by expressing the polynomial $p(x)$ in terms of higher degree ($> d$) Bernstein base polynomials.

Example 3: Fig. 3 shows the polynomial $p(x) = x^3 - 5x^2 + 2x + 4$ from the previous example, the terms $b_i^3 B_i^3(x)$ of its Bernstein form and the constants b_i^3. On the interval $[0, 1]$, the polynomial is bounded by the minimal and maximal Bernstein coefficients, $b_3^3 = 2$ and $b_1^3 = 14/3$. The first of these coefficients is sharp; the second is not.

To compute the Bernstein coefficients b_i^d from the power form coefficients a_i, we write the point x on the interval $[0, 1]$ in terms of its barycentric coordinates, $x = \alpha_0 v_0 + \alpha_1 v_1$, with $\alpha_0, \alpha_1 \geq 0$ and $\alpha_0 + \alpha_1 = 1$ and where $v_0 = 0$ and $v_1 = 1$ are the vertices of the interval $[0, 1]$. We see that $\alpha_1 = x$ and $\alpha_0 = 1 - x$ and that the Bernstein base polynomials (18) are homogeneous polynomials of degree d in α_0 and α_1. To write $p(x)$ (16) as a homogeneous polynomial in α_0 and α_1, we simply substitute $x = \alpha_0 0 + \alpha_1 1 = \alpha_1$ and multiply each degree-i homogeneous component of $p(\alpha_0, \alpha_1)$ ($i \leq d$) by $1 = (\alpha_0 + \alpha_1)^{d-i}$, i.e.,

$$\begin{aligned} p(\alpha_0, \alpha_1) &= \sum_{i=0}^{d} a_i \alpha_1^i (\alpha_0 + \alpha_1)^{d-i} \\ &= \sum_{i=0}^{d} a_i \alpha_1^i \left(\sum_{j=0}^{d-i} \binom{d-i}{j} \alpha_0^{d-i-j} \alpha_1^j \right) \\ &= \sum_{k=0}^{d} \left(\sum_{i=0}^{k} a_i \binom{d-i}{k-i} \right) \alpha_1^k \alpha_0^{d-k}. \end{aligned}$$

Comparing with (17) and noting that

$$B_k^d(x) = B_k^d(\alpha_0, \alpha_1) = \binom{d}{k} \alpha_1^k (\alpha_0)^{d-k} \tag{19}$$

we obtain

$$b_k^d = \sum_{i=0}^{k} \frac{\binom{d-i}{k-i}}{\binom{d}{k}} a_i = \sum_{i=0}^{k} \frac{\binom{k}{i}}{\binom{d}{i}} a_i$$

where the last equality follows from the identity $\binom{d-i}{k-i}\binom{d}{i} = \binom{d}{k}\binom{k}{i}$.

Bounds on the values attained by a polynomial over an arbitrary interval $[a, b]$ can be obtained using essentially the same technique by writing $x = \alpha_0\, a + \alpha_1\, b$ and computing coefficients $b_k^{[a,b],d}$ with respect to the basis in (19).

B. Bernstein Expansion Over a Parametric Convex Polytope

In this subsection, we extend the Bernstein expansion technique to multivariate parametric polynomials defined over parametric polytopes of any dimension. This extension also generalizes the so-called Bernstein-Bezier form of a polynomial defined over a triangle [28]. We are given a (rational) multivariate polynomial

$$p(x_1, x_2, \ldots x_n) = \sum_{i_1=0}^{d_1} \sum_{i_2=0}^{d_2} \cdots \sum_{i_n=0}^{d_n} a_{i_1,i_2,\ldots,i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \tag{20}$$

where the coefficients $a_{\mathbf{i}}$ may also themselves be polynomials in the parameters $\mathbf{q}$, i.e., $a_{\mathbf{i}}(\mathbf{q}) \in \mathbb{Q}[\mathbf{q}]$

$$a_{\mathbf{i}}(\mathbf{q}) = \sum_{j_1=0}^{m_1} \sum_{j_2=0}^{m_2} \cdots \sum_{j_r=0}^{m_r} b_{j_1,j_2,\ldots,j_r} q_1^{j_1} q_2^{j_2} \cdots q_r^{j_r}$$

and $p(\mathbf{x}) \in (\mathbb{Q}[\mathbf{q}])[\mathbf{x}]$. This parametric polynomial is defined over some $\mathcal{V}$-parametric polytope $P(\mathbf{q})$ (9), where the generators $\mathbf{v}_i(\mathbf{q}) \in \mathbb{Q}[\mathbf{q}]$ may be arbitrary polynomials in the parameters $\mathbf{q}$. Recall that some of these generators may be vertices for only a subset of the values of the parameters.

To compute lower and upper bounds on this polynomial $p(\mathbf{q})(\mathbf{x})$, over the polytope $P(\mathbf{q})$, we essentially follow the procedure from the previous section. We first write $\mathbf{x}$ as a convex combination of the vertices $\mathbf{x} = \sum_i \alpha_i \mathbf{v}_i(\mathbf{q})$ and substitute this expression in the polynomial $p(\mathbf{q})(\mathbf{x})$. We then multiply each term in the result with the appropriate power of $1 = \sum_i \alpha_i$ to obtain a homogeneous polynomial in the α_i of degree d, where d is the maximum of the d_i. Finally, we compute the coefficients $b_{\mathbf{k}}^{P,d}(\mathbf{q})$, for $\mathbf{k} = (k_1, \ldots, k_n)$, $0 \leq k_i$, $\sum k_i = d$, in terms of the *generalized Bernstein base polynomials* $B_{\mathbf{k}}^d$. These generalized Bernstein base polynomials are the terms in the expansion of

$$\begin{aligned} 1 &= (\alpha_1 + \alpha_2 + \cdots + \alpha_n)^d \\ &= \sum_{\substack{k_1,k_2,\ldots,k_n \geq 0 \\ k_1+k_2+\cdots+k_n=d}} \binom{d}{k_1, k_2, \ldots, k_n} \alpha_1^{k_1} \alpha_2^{k_2} \cdots \alpha_n^{k_n} \\ &= \sum_{\substack{k_1,k_2,\ldots,k_n \geq 0 \\ k_1+k_2+\cdots+k_n=d}} B_{\mathbf{k}}^d(\boldsymbol{\alpha}), \end{aligned}$$

where

$$\binom{d}{k_1, k_2, \ldots, k_n} = \frac{d!}{k_1! k_2! \ldots k_n!} \tag{21}$$

are the multinomial coefficients. As before, the $B_{\mathbf{k}}^d(\boldsymbol{\alpha})$ are nonnegative on $P(\mathbf{q})$ and sum to 1 and so can be considered to be the coefficients in the expression of $p(\mathbf{q})(\mathbf{x})$ as a convex combination of the $b_{\mathbf{k}}^{P,d}(\mathbf{q})$. We therefore have

$$\min_{\substack{k_1,k_2,\ldots,k_n \geq 0 \\ k_1+k_2+\cdots+k_n=d}} b_{\mathbf{k}}^{P,d}(\mathbf{q}) \leq p(\mathbf{q})(\mathbf{x}) \leq \max_{\substack{k_1,k_2,\ldots,k_n \geq 0 \\ k_1+k_2+\cdots+k_n=d}} b_{\mathbf{k}}^{P,d}(\mathbf{q}) \tag{22}$$

on the parametric polytope $P(\mathbf{q})$. Note that the generalized Bernstein base polynomials we use here are different from

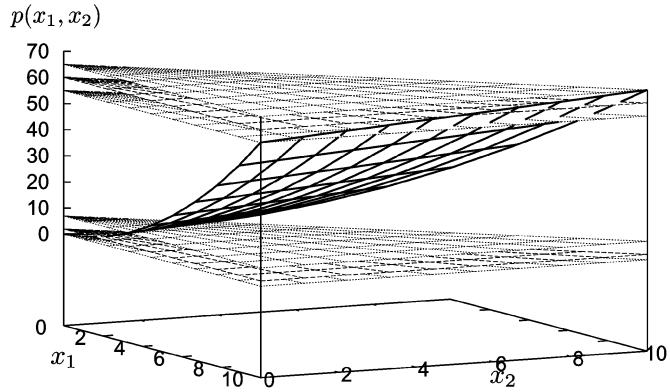

Fig. 4. The polynomial $p(x_1, x_2) = 1/2x_1^2 + 1/2x_1 + x_2$ and the corresponding Bernstein coefficients.

the multivariate Bernstein polynomials [8], [29], which are products of standard Bernstein polynomials.

Example 4: Consider the polynomial $p(x_1, x_2) = 1/2x_1^2 + 1/2x_1 + x_2$ over the parametric polytope generated by the points $(0 \quad 0)^t$, $(N \quad 0)^t$ and $(N \quad N)^t$. Hence any point $(x_1 \quad x_2)^t$ in the polytope is a convex combination of these points, $(x_1 \quad x_2)^t = \alpha_1 (0 \quad 0)^t + \alpha_2 (N \quad 0)^t + \alpha_3 (N \quad N)^t$, with $0 \leq \alpha_i \leq 1$ and $\sum_{i=1}^{3} \alpha_i = 1$. By replacing $(x_1 \quad x_2)^t$ with this convex combination, a new polynomial is obtained whose variables are α_1, α_2, α_3:

$$\frac{1}{2}N^2\alpha_2^2 + N^2\alpha_2\alpha_3 + \frac{1}{2}N^2\alpha_3^2 + \frac{1}{2}N\alpha_2 + \frac{3}{2}N\alpha_3.$$

Monomials of degree less than 2 are transformed into sums of monomials of degree 2 through multiplication by $(\alpha_1+\alpha_2+\alpha_3)$. The final polynomial is

$$p(\alpha_1, \alpha_2, \alpha_3) = \left(\frac{1}{2}N^2 + \frac{1}{2}N\right)\alpha_2^2 + \left(\frac{1}{2}N^2 + \frac{3}{2}N\right)\alpha_3^2 + \frac{1}{2}N\alpha_1\alpha_2 + \frac{3}{2}N\alpha_1\alpha_3 + (N^2 + 2N)\alpha_2\alpha_3.$$

The basis is built from the expansion of $(\alpha_1 + \alpha_2 + \alpha_3)^2$ providing the following monomials:

$$B_{2,0,0} = \alpha_1^2 \quad B_{0,2,0} = \alpha_2^2 \quad B_{0,0,2} = \alpha_3^2$$
$$B_{1,1,0} = 2\alpha_1\alpha_2 \quad B_{1,0,1} = 2\alpha_1\alpha_3 \quad B_{0,1,1} = 2\alpha_2\alpha_3.$$

Rewriting $p(\alpha_1, \alpha_2, \alpha_3)$ in terms of this basis, we obtain

$$0\, B_{2,0,0} + \left(\frac{1}{2}N^2 + \frac{1}{2}N\right)B_{0,2,0} + \left(\frac{1}{2}N^2 + \frac{3}{2}N\right)B_{0,0,2} + \frac{1}{4}NB_{1,1,0} + \frac{3}{4}NB_{1,0,1} + \left(\frac{1}{2}N^2 + N\right)B_{0,1,1}.$$

It can then be concluded that the polynomial varies between 0 and $1/2N^2 + 3/2N$. Since both of these coefficients are sharp coefficients, the bounds are exact bounds. The graph of the polynomial and the corresponding Bernstein coefficients are shown in Fig. 4 for $N = 10$.

V. Removing Redundant Bernstein Coefficients

The previous section showed how to compute *all* Bernstein coefficients of a polynomial defined over a polytope. The Bernstein coefficients themselves are also polynomials defined over a polytope (of lower dimension). Many of these coefficients will typically be redundant with respect to the computation of a lower or upper bound, in the sense that there are other polynomials that always yield smaller (or larger) values over the whole domain.

The simplest way to eliminate redundant Bernstein coefficients, is to examine the sign of the difference between two polynomials. If the sign is constant over the domain (where a zero sign may be treated as either positive or negative), then one of the two is redundant. Some easy ways of determining the sign of a (difference) polynomial are as follows.

1) If the difference is a constant, the check is trivial.
2) If the difference is linear in the parameters, we add the constraint that the difference be strictly larger than zero to the domain and check whether the resulting domain is empty. For example, in (14), the polynomial $n^2/2 + 3/2$ is redundant since
$$\left(\frac{n^2}{2} + \frac{3}{2}\right) - \left(\frac{n^2}{2} + \frac{n}{2} + 1\right) = \frac{1}{2} - \frac{n}{2}$$
and this difference polynomial is never greater than zero for $n \geq 1$. The polynomial $n^2/2 - n/2 + 2$ is eliminated for the same reason, while $n^2 - n/4 + 5/4$ and $n^2 - 3n/4 + 7/4$ are eliminated because they are redundant with respect to $n^2 + 1$. If it turns out that the sign of the difference varies over the domain, we could in principle further subdivide the domain along the above constraint.
3) If the domain over which we want to determine the sign is bounded, we can apply Bernstein expansion again on the difference over this domain, which is now considered to be a fixed domain without parameters. The resulting Bernstein coefficients are therefore constants. If all the non-zero Bernstein coefficients have the same sign, then so will the difference over the whole domain. For example, if we assume that there is an upper bound on n, say 1000, then we can perform Bernstein expansion on
$$\left(\frac{n^2}{2} + \frac{n}{2} + 1\right) - \left(n^2 + 1\right) = -\frac{n^2}{2} + \frac{n}{2} \tag{23}$$
over $1 \leq n \leq 1000$. The resulting Bernstein coefficients are $\{0, -499500, -999/4\}$ and so we can conclude that $n^2/2 + n/2 + 1$ is redundant with respect to $n^2 + 1$. Note that if the polynomial is univariate of degree d with coefficients c_i then we know that all real roots lie in the interval $[-M, M]$ with $M = 1 + \max_{0 \leq i \leq d-1} |c_i|/|c_d|$ (Cauchy's bound). It is therefore sufficient to consider the intersection of a strict superset of this interval with the possibly unbounded domain of interest. In the example, it would be sufficient to consider the domain $1 \leq n \leq 3$.
4) If the domain over which we want to determine the sign is not bounded, but there is a lower bound on one of the parameters, we can write the Taylor expansion of the difference about this lower bound and determine the signs of

the coefficients in the Taylor expansion. Note that we can easily compute these coefficients using synthetic division. If all signs are constant and equal, then also the difference will have this constant sign. For example, we can write (23) as

$$-\frac{1}{2}(n-1)-\frac{1}{2}(n-1)^2$$

and the coefficients are clearly negative, so we can again conclude that $n^2/2+n/2$ is redundant, over the whole domain $n \geq 1$.

In general, we will however not be able to identify all but one polynomial as redundant. Still, it may be desirable in some cases to have only a single polynomial associated to every subdomain, such that for a given subdomain only this single polynomial needs to be evaluated. If the difference between two polynomials is linear then this can easily be accomplished by splitting the domain along the hyperplane where the difference is zero. For example, suppose we have two polynomials $n^2+3n-500$ and n^2+n in the maximum expression associated to the domain $n \geq 4$. The difference between these two polynomials $2n-500$ is zero along $n=250$ and so we would split the domain into, say, $4 \leq n < 250$ and $250 \leq n$. If the differences between pairs of polynomials is not linear, but they are univariate, then we may not be able to easily split the domain into subdomains where only a single polynomial remains, but based on Cauchy's bounds, we can identify and split off a region of "big" values where the upper bound is given by a single polynomial.

VI. Incremental Computation

In Section IV-B, we have shown how to apply Bernstein expansion to an arbitrary parametric polytope, specified as a convex combination of parametric points (9). The complexity of this procedure may be quite high, however, as the number of Bernstein coefficients that appear in (22) is equal to the number of multinomial coefficients (21), which is

$$\binom{d+N-1}{d} > \frac{N^d}{d!}$$

where d is the degree of the polynomial and N is the number of vertices. Furthermore, if the parametric polytope is specified through a set of (linearly) parametric linear constraints (10), then by McMullen's Upper Bound Theorem [30], the number of vertices is $N=O(m^{\lfloor n/2 \rfloor})$, where m is the number of constraints and n is the dimension of the polytope.

To alleviate this computational complexity, we may apply Bernstein expansion *incrementally*. We will focus on *linearly* parametrized polytopes here. Although the technique below could in principle also be applied to non-linearly parametrized polytopes, the required operations are more expensive in this setting, suggesting that the direct (non-incremental) application may be more appropriate for these problems. In a first step of the incremental approach, we consider all but one variable to be an extra parameter and compute the parametric vertices of this parametric interval with the corresponding chamber decomposition. In each of the chambers, we then compute the Bernstein coefficients. Note that the single variable that was not considered to be a parameter in this computation, no longer appears in these Bernstein coefficients. The above procedure is then applied recursively until all variables have been eliminated.

In the incremental computation, all parametric polytopes are intervals. They therefore have 2 vertices and require only $d+1$ Bernstein coefficients. On the other hand, the interval Bernstein expansion has to be performed in all nodes of a tree with $2n$ levels, where n is the number of variables, that branches alternately over the chambers and the Bernstein coefficients. Furthermore, the final number of chambers may be larger than the number of chambers obtained through a direct application of Bernstein expansion. However, this possible increase in the number of chambers may improve the accuracy. More importantly, we can remove redundant Bernstein coefficients, as explained in Section V, at each (other) level in the computation, which typically leads to a huge reduction in the total number of Bernstein coefficients considered, for high dimensional polytopes with many vertices. For low-dimensional polytopes with a small number of vertices, the direct (non-incremental) approach will be faster. Note that we only apply the technique based on Bernstein expansion for removing redundant coefficients (technique 3 of Section V) in leaves of the tree.

Using an incremental computation, we can also handle the parametric boxes of Clauss *et al.* [8]. These boxes have the general form

$$[l_1(\mathbf{q}),u_1(\mathbf{q})]\times[l_2(x_1,\mathbf{q}),u_2(x_1,\mathbf{q})]\times\cdots \times[l_n(x_1,\ldots,x_{n-1},\mathbf{q}),u_n(x_1,\ldots,x_{n-1},\mathbf{q})]$$

where the lower and upper bound l_i, u_i are polynomials in the parameters *and* the preceding variables. In general, these parametric boxes cannot be represented as parametric polytopes of types (9) or (10). Clauss *et al.* [8] use product Bernstein base polynomials over the unit cube $[0,1]^n$ in their Bernstein expansion, after a linear transformation

$$y_i=\frac{x_i-l_i(x_1,\ldots,x_{i-1},\mathbf{q})}{u_i(x_1,\ldots,x_{i-1},\mathbf{q})-l_i(x_1,\ldots,x_{i-1},\mathbf{q})}$$

of the parametric box to this unit cube. Not only does this approach require a special handling of the roots of the denominators in the linear transformation, these roots need to be *detected* first, which may be non-trivial if the lower and upper bounds are not linear.

Instead of transforming a parametric box to the unit cube, we can treat a parametric box as a parametric interval in the last coordinate x_n while all the other variables $\hat{\mathbf{x}}=(x_1,\ldots,x_{n-1})$ are considered to be extra parameters. The application of symbolic Bernstein expansion on this coordinate yields

$$\min_{0\leq i\leq d} b_i^d(\hat{\mathbf{x}},\mathbf{q}) \leq p(\hat{\mathbf{x}},\mathbf{q})(x_n) \leq \max_{0\leq i\leq d} b_i^d(\hat{\mathbf{x}},\mathbf{q}).$$

This procedure is then applied recursively on each (non redundant) Bernstein coefficient $b_i^d(\hat{\mathbf{x}},\mathbf{q})$ over the box

$$[l_1(\mathbf{q}),u_1(\mathbf{q})]\times[l_2(x_1,\mathbf{q}),u_2(x_1,\mathbf{q})]\times\cdots \times[l_{n-1}(x_1,\ldots,x_{n-2},\mathbf{q}),u_{n-1}(x_1,\ldots,x_{n-2},\mathbf{q})].$$

Note that by not insisting on mapping any interval to the unit interval, we avoid any problems associated to possible roots in the denominators of the corresponding linear transformation. Also note that neither our incremental approach or the approach of [8] makes any distinction between the lower and the upper bound of a coordinate. In applications where the domain is defined to be empty when the upper bound is smaller than the lower bound, we can improve the accuracy by disregarding, at each level, the chambers where the upper bound is known to be smaller than the lower bound.

VII. Software Implementation

We have implemented the computation of a bound on an arbitrary multivariate polynomial defined over a linearly parametrized convex polytope, as explained in Section III-B, in our Bernstein library. This includes the computation of the Bernstein coefficients of the polynomial as well as the removal of redundant coefficients. Our library is built on top of two other libraries:

- the polyhedral library `PolyLib` [25] to compute the vertices of a linearly parametrized polytope,
- the `GiNaC` library [31] for symbolic polynomial manipulations.

Both of these libraries use the GMP library (as part of the `CLN` library [32] in the case of `GiNaC`) for arbitrary precision arithmetic on integers. Furthermore, the `bernstein` library has been integrated into the `barvinok` library [33], which has been augmented with a procedure for computing a bound on the result of a counting problem using bernstein. This procedure implements the approach discussed in Section III-A and supports the incremental computation of Section VI.

VIII. Applications to Memory Requirement Estimation

In this section, we describe some applications to memory requirement estimation. In each of these applications we are given a polynomial expression of the amount of "memory in use" at a given "execution point" and we want to compute an upper bound on the amount of memory used over all execution points. The memory in use can be the set of live array elements, the tokens in a FIFO, the elements accessed between two uses of the same element or the size of the memory scope of a method in terms of its parameter values. Our technique can also be used to extend the applicability of the applications of [8] from "boxes" to more general parametric polytopes. We will not repeat those applications here.

A. Memory Size Computation

The problem of computing the "exact memory size" of a program is that of finding the minimum amount of memory locations needed to store the data of the program during its execution [6]. This problem is basically the liveness analysis we used as an example in Section II and variations of this problem have been studied earlier in the literature (e.g., [1]–[4]). Zhu *et al.* [6] distinguish themselves from previous research by computing the memory size exactly, rather than approximately. We focus on their work because it is the most recent and because they cite some numbers to which we can compare our results. They propose a rather complicated algorithm where they first decompose the array references into disjoint linearly bounded lattices and then compute the number of live elements both between consecutive top-level loops and inside top-level loops. For this last computation, they determine the iteration where the number of live elements changes and then presumably iterate over all these iterations to find the maximum number of live elements. Their algorithm is fundamentally non-parametric, so they need to redo the whole computation for each value of the parameters.

Using Bernstein expansion, we can compute (an upper bound of) the memory size parametrically. We implemented a very straightforward algorithm based on the simple idea that the number of live elements increases by one when an array element is written or read that will be read (again) later and decreases by one when an array element is read that was written or read before. Intermediate reads satisfy both conditions and do therefore not alter the number of live elements. The first step is the computation of pairs of consecutive accesses to array elements, where the first is either a write or a read and the second is a read. In particular, for each read, we compute the last access (read or write) to the same memory element. (Since we compute the last access, we do not require a one-to-one mapping between the iteration domains and the array space.) If all loop bounds, conditions and index expressions are affine combinations of outer loop iterators and symbolic parameters, this computation can be performed using a variation of array dataflow analysis [34], resulting in a union of relations between iterations of two statements described by linear constraints.

The domains of these relations correspond to increases in the number of live elements, while the ranges correspond to decreases. For each iteration of each statement, we then compute both the number of elements in any of these domains that precede the given iteration and the number of elements in any of the ranges that precede the given iteration. Both computations are performed using barvinok [33]. Taking the difference, we obtain the number of live elements at the given iteration, as a piecewise (quasi-)polynomial in the iterators. The maximum number of live elements is then computed as explained in Section III.

The main limitation is the fact that the array dataflow analysis requires all constraints to be affine. This restriction can be relaxed to some extent through the use of fuzzy array dataflow analysis [35], but the resulting relations may then also contain non-affine constraints, while the computation of the number of integer points (in the domains and ranges of these relations) does not handle such constraints in general.

Although the procedure outlined above can still be significantly optimized by avoiding redundant computations, even the straightforward implementation can compute the parametric memory size for the 2D Gaussian blur filter in 21 seconds on an Athlon MP 1500+ with 512MiB internal memory, while Zhu [36] reports computation times of 3 and 103 seconds on a slightly faster machine for parameter values $N = 100$, $M = 50$ and $M = N = 500$ respectively. The size we compute is $MN + 5$, which agrees with the values 5005 and 250005 reported by [36]. During the computation of this bound, we place a lower bound (of, say, 10) on the parameters. Without this lower bound, the algorithm would end up computing

specialized values for various small values of the parameters, significantly increasing the computation time without providing any interesting additional information.

We should also point out that the algorithm of [6] appears to be fairly inefficient for large values of the parameters. In an alternative, again very straightforward, implementation, we first basically perform the dependence analysis outlined above and generate code using CLooG [37] to count the number of live elements by incrementing a counter each time a value is read or written that is still needed and decrementing the same counter each time a value is read and report the maximal value attained by the counter. For each value of the parameters we then compile and execute the generated code. For the same application, we found that the analysis and code generation takes about 9 seconds, compilation takes about 0.5 seconds and the actual execution is too fast to be measured for $N = 100$, $M = 50$ while it takes about 0.03 seconds for $M = N = 500$.

Note that the size computed by our procedure may be an overestimate. However, the actual memory size may not be very useful, since in order to fit all data in the "exact memory size" you would still have to derive an appropriate mapping of the array elements to this minimally sized memory. This addressing issue is not discussed in [6].

B. Computing FIFO Sizes in Process Networks

The conversion of a sequential program to a process network is a way of exposing the task-level parallelism in the program [38]. In a process network, independent processes communicate with each other through communication channels. The derivation of process networks is an extension of array dataflow analysis [34], where array reads are analyzed to determine where the data was produced and where all array accesses are subsequently replaced by reads and writes to the communication channels. In many cases, the reads and writes occur (or can be made to occur) in the same order and the communication channel is a FIFO. In an idealized form, these FIFOs are unbounded, but for a practical (hardware or software) implementation we need to be able to compute bounds on the sizes of the FIFOs.

To compute the maximal number of tokens in a self-loop FIFO, we again apply the technique of Section III. For FIFOs between distinct processes, the process network needs to be scheduled first to fix the relative execution times of the otherwise independent processes. Note that this schedule is only used for the purpose of computing FIFO sizes that allow for a deadlock-free execution, and not during the actual execution of the network. For more information and an example, we refer to [39].

C. Reuse Distances

The (forward) reuse distance of a memory access to a memory element is the number of distinct memory elements accessed between the given access and the next access to the same memory element. It is a measure for the locality of the access as the element will still be in the cache on the next access depending on whether the reuse distance is smaller than the cache size, assuming that the cache is fully associative with LRU replacement policy [40]. Beyls *et al.* [40] propose to use the reuse distance to select cache hints. Since the cache hint of a given instruction is fixed during the entire execution of the program, while it may give rise to many accesses with different reuse distances, they propose to base their cache hint selection on the cache that is sufficiently large to hold 90% of the elements accessed by the instruction until their next use.

To be able to determine the appropriate cache size, they need to *evaluate* the reuse distance for each loop iteration of the loops surrounding the given instruction, even though the reuse distance itself can be computed parametrically in terms of the loop iterators and structural parameters. Although Bernstein expansion cannot help to easily determine the minimum size that will hold 90% of the accesses, it can help to determine the minimum size that will hold all accesses that will still be reused, by computing an upper bound of the reuse distance over all iterations. This strategy can be further refined by also considering the lower bound of the reuse distance, which can be computed in a similar way or simply by noticing that $\min f(\mathbf{i}) = -\max -f(\mathbf{i})$, as well as the average reuse distance, which can also be computed parametrically [41].

D. Estimating Dynamic Memory Requirements

In [42], Braberman *et al.* present a technique for computing a parametric upper-bound of the amount of memory dynamically *requested* by Java-like imperative programs. The idea consists in quantifying dynamic allocations performed by a method. Given a method m with parameters P_m the authors provide an algorithm that computes a polynomial over P_m which over-approximates the amount of memory allocated during the execution of m. This bound is a symbolic over-approximation of the total amount of memory the application *requests* to a virtual machine via new statements, but not the *actual* amount of memory really consumed by the application. This is because memory freed by the GC is *not* taken into account. Roughly speaking, the technique identifies allocation sites (new statements) reachable from the method under analysis and counts the number of times those statements are visited. To do that, linear invariants describing possible valuations of variables at each allocation site are generated and the number of integer solutions of those invariants are counted. Finally the result is adapted to consider the type of each allocated object. For region-based memory management [43] where objects are placed in regions associated with computation units, the same technique allows obtaining polynomial bounds of the size of every memory region, assuming regions are synthesized at compile-time (e.g., [43], [44]).

Given a method m with parameters P_m it is posible to obtain a parametric upper-bound of the amount of memory necessary to *safely* execute m and all methods it calls, without running out of memory. To compute this estimation an ideal memory manager is approximated using a region-based memory management where region lifetime is associated with method lifetime. Then, in order to obtain an expression of the peak consumption of the method under analysis all the potential configurations of region stacks at run-time are approximated by considering the sum of the regions asociated with all potential call chains in the application call graph. This modeling leads to a set of polynomial (and parametric) maximization problems. One of them involves computing the largest region sizes for a set of calling contexts. Since synthesized region sizes are polynomials and

calling contexts can be modeled using parametric domains, we can solve this problem using the technique proposed here. In general, the solution may still contain maximum expressions over a finite set of polynomials, which will need to be evaluated at run-time when the values of the parameters are known. More details can be found in [45].

IX. Related Work

A. Handling Polynomials in Program Analysis

Maslov and Pugh [46] present a technique to simplify polynomial constraints. It is based on a decomposition of a polynomial constraint into a conjunction of affine constraints and 2-variable hyperbolic and elliptical inequalities and equalities that can later be linearized. Hence their approach is not general and can only handle those polynomials that can be decomposed in this way.

Blume and Eigenmann [47] present an algorithm for determining the sign of a symbolic expression. It is assumed that each variable has a (symbolic) lower and upper bound and these ranges are repeatedly substituted in the expression until a non-negative constant lower bound or a non-positive constant upper bound is found on the expression. In each iteration, the expression is simplified using a set of rewrite rules. If a variable occurs multiple times in the same expression, then overly conservative bounds can be generated. However, if they can determine, by recursively applying their algorithm to the first order forward difference of the polynomial, that the expression is monotonically non-increasing or non-decreasing in a given variable, then they can safely substitute the lower and upper bounds of the variable simultaneously in the whole expression, leading to a tighter bound. Although this technique was only intended for determining the sign of a symbolic expression, it can also be used to find a symbolic bound by simply not substituting some of the variables. The main disadvantages of this technique are that it only works over "boxes" and that the accuracy can be very low for non-monotonic expressions. See Section X

Van Engelen *et al.* [48] apply the same (recursive) monotonicity test of [47] in the specific context of bounding a polynomial over the parametric interval $[0, n-1]$. They compute the forward difference ($h(i)$ in their Lemma 3.1) in a slightly different way, though. In particular, they use the Newton series representation of a polynomial, which expresses the polynomial in the "falling factorial" basis. This representation allows them to more cheaply detect some cases of monotonic polynomials. Similarly to [47], they resort to value range analysis if the polynomial is not recursively monotonic.

Bernstein expansion has already been used by [8] to handle parametric polynomials defined over nested parametric intervals called parametric boxes. Several applications such as non-linear dependence analysis or dead code elimination are shown. However, the proposed approach is limited to domains defined as boxes. These boxes also need to be linearly transformed to unit boxes. This transformation, when applied to parametric boxes, has to exclude some parameter values for which the transformation would yield divisions by zero. Hence the considered polynomials have to be evaluated for these specific values. These drawbacks are entirely removed with the method presented in this paper.

B. Memory Requirement Estimation

1) Static Memory: The problem of finding (an approximation) of the minimal amount of memory required to run a program given a schedule has been studied before [1]–[4], [6]. However, most authors make simplifying assumptions, such as uniformly generated [4], or the assumption that the number of live elements is an affine expression of the iterators [3], rather than the more general case of a polynomial. The techniques of other authors (e.g., [6]) only work for non-parametric programs. Kjeldsberg *et al.* [5] estimate the memory requirements when the execution order is only partially known. Many authors have also considered the problem of finding good memory mappings. We refer to [49] for an overview and a mathematical framework for handling this problem.

2) Dynamic Memory: The problem of dynamic memory estimation has been studied for functional languages in [50]–[52]. The work of Hofmann and Jost [50] statically infers, by type derivation and linear programming, linear expressions that depend on function parameters. The technique is stated for functional programs running under a special memory mechanism (free list of cells and explicit deallocation in pattern matching). The computed expressions are linear constraints on the sizes of various parts of data. Hughes [51] propose a variant of ML together with a type system based on the notion of sized types, such that well typed programs are proven to execute within the given memory bounds. The technique proposed by Unnikrishnan [52] consists in, given a function, constructing a new function that symbolically mimics the memory allocations of the former. The computed function has to be executed over a valuation of parameters to obtain a memory bound for that assignment. The evaluation of the bound function might not terminate, even if the original program does.

For imperative object-oriented languages, a method has been proposed by Chin *et al.* [53] which relies on a type system and type annotations, similar to [51]. It does not actually synthesize memory bounds, but statically checks whether size annotations (Presburger formulas) are verified. It is therefore up to the programmer to state the size constraints, which are moreover required to be linear.

X. Experiments

As we are unaware of any other publicly available implementation of an algorithm for computing bounds on a polynomial over a polytope, we also implemented a variation on range propagation to compare our Bernstein based technique against. In each step of this range propagation, we eliminate one of the variables in terms of the remaining variables and the parameters. Note that the bounds on the variables are given by linear constraints. This means that a lower bound on x in terms of y is also an upper bound on y in terms of x. It therefore does not make sense in our case to topologically sort the variables as proposed in [47]. As the selected variable may have more than one lower or upper bound, the domain is first split into parts where it has only one pair of bounds, as in the incremental computation of Section VI.

When substituting a variable by its range in a multiplication, it is crucially important to know the signs of both factors, as

TABLE I
EXPERIMENTAL COMPARISON BETWEEN RANGE PROPAGATION AND BERNSTEIN EXPANSION

program		propagation		Bernstein	
	# problems	error	size	error	size
Example from Section II					
	1	2.28	7	0.01	4
FIFO size computation (optimized)					
LU-Factor	36	0	273	0	248
QR-Decomp	2	0	4	0	4
Faddeev	14	0	30	0	28
Gauss-Elim.	14	1.06	52	0.09	49
Motion Est.	38	0	76	0	76
FIFO size computation (unoptimized)					
LU-Factor	84	0.04	1328	0	1120
QR-Decomp	2	0	4	0	4
Faddeev	18	0.06	38	0	36
Gauss-Elim.	16	1.36	88	0.11	78
Motion Est.	69	0	138	0	138
Memory size computation (Section VIII-A)					
gauss	7	0	18	0	14
durbin	19	0.01	18	0.01	18
dynprog	4	0.96	14	0	8

otherwise we lose all information. We therefore choose to evaluate the given expressions as an expanded polynomial in all the variables, i.e., as a sum of monomials, and split the domain into orthants, fixing the sign of each monomial. Before we substitute a variable x_1 in the polynomial $p(x_1, \hat{\mathbf{x}}, \mathbf{q})$, we first check if the sign of $p(x_1+1, \hat{\mathbf{x}}, \mathbf{q}) - p(x_1, \hat{\mathbf{x}}, \mathbf{q})$ is constant by applying this technique recursively as a non-parametric problem. If this sign is constant, then $p(x_1, \hat{\mathbf{x}}, \mathbf{q})$ is monotonically increasing or decreasing and we replace x_1 in the whole polynomial by its upper or lower bound. Otherwise we replace x_1 by its upper bound in the positive monomials and by its lower bound in the negative monomials, and vice versa when we want to compute a lower bound.

To measure the accuracy of the methods, we compute the relative error on the range, i.e., upper bound minus lower bound plus one, computed using both methods. In particular, we compute the average relative error over the first 50 values of the parameters. To measure the size of the output, we count the number of polynomials in the final result.

The results on some small to medium-sized problems are shown in Table I. The first column identifies the source code on which one of the applications from Section VIII was applied. For the FIFO size computation of Section VIII-B, we show two sets of the results. The first set is for an optimized computation where we exploit the fact that in this application, the upper bound can only be reached in the first iteration of an inner loop. The second set shows the results of the computations without this optimization. The second column shows the number of piecewise polynomials for which a bound was computed. Note that each of these piecewise polynomials may have several cases, resulting in the computation and combination of bounds over several polytopes. The remaining columns show the relative error and the number of resulting polynomials for both methods, as explained above.

The table shows that Bernstein expansion never performs worse than range propagation and sometimes performs significantly better. The data sets for which both methods give exact results are those for which the monotonicity test applies. It should be noted here that the applications typically require only the upper bound, and that the accuracy of the upper bound may be better (or worse) than the accuracy of the range. However, it is difficult to define a proper measure for the accuracy of the upper bound on its own. The number of polynomials in the output is usually smaller for Bernstein expansion as this technique is able to determine more signs of differences between polynomials.

XI. CONCLUSION

Memory requirement evaluation of applications is a major issue in the design of computer systems, and specifically in the case of embedded systems. We have shown for several application examples that this problem can often consist in maximizing a parametric and multivariate polynomial defined over a parametric convex domain. We proposed an original approach based on Bernstein expansion to compute accurate bounds for such polynomials, and even exact bounds in some cases. It has been implemented and is freely available.

We have also shown that static analysis of programs can provide high quality results for complicated and critical issues as soon as efficient mathematical tools are well used and adapted. Static analysis is superior to dynamic and experimental approaches, such as profiling or iterative compilation, since it can directly provide accurate and correct results. Moreover, these results, when parametrized, can cover all possible execution configurations from only one unique program analysis process.

In this paper, Bernstein expansion is used to analyze polynomials resulting from previous program analysis steps. However, Bernstein polynomials can also be used to perform polynomial interpolation. We are currently investigating the use of Bernstein interpolation to model data that is too difficult or too complex to be handled directly, enabling some new interesting program transformations.

REFERENCES

[1] I. M. Verbauwhede, C. J. Scheers, and J. M. Rabaey, "Memory estimation for high level synthesis," in *Proc. 31st Annu. Conf. Design Automation (DAC'94)*, New York, NY, 1994, pp. 143–148.

[2] P. Grun, F. Balasa, and N. Dutt, "Memory size estimation for multimedia applications," in *Proc. 6th Int. Workshop on Hardware/Software Codesign (CODES/CASHE'98)*, Washington, DC, 1998, pp. 145–149.

[3] Y. Zhao and S. Malik, "Exact memory size estimation for array computations," *IEEE Trans. Very Large Scale Integrat. (VLSI) Syst.*, vol. 8, no. 10, pp. 517–521, Oct. 2000.

[4] J. Ramanujam, J. Hong, M. T. Kandemir, and A. Narayan, "Reducing memory requirements of nested loops for embedded systems," in *Proc. Design Automation Conf.*, 2001, pp. 359–364.

[5] P. G. Kjeldsberg, F. Catthoor, and E. J. Aas, "Storage requirement estimation for optimized design of data intensive applications," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 9, no. 2, pp. 133–158, 2004.

[6] H. Zhu, I. I. Luican, and F. Balasa, "Memory size computation for multimedia processing applications," in *Proc. 2006 Conf. Asia-South Pacific Design Automation (ASP-DAC'06)*, New York, NY, 2006, pp. 802–807.

[7] P. Feautrier, *The Data Parallel Programming Model*. New York: Springer-Verlag, 1996, vol. LNCS 1132, ch. Automatic Parallelization in the Polytope Model, pp. 79–100.

[8] P. Clauss and I. Tchoupaeva, E. Duesterwald, Ed., "A symbolic approach to Bernstein expansion for program analysis and optimization," in *Proc. 13th Int. Conf. Compiler Construction (CC 2004)*, Apr. 2004, vol. 2985, pp. 120–133.

[9] J. A. D. Loera, R. Hemmecke, M. Köppe, and R. Weismantel, "Integer polynomial optimization in fixed dimension," *Math. Oper. Res.*, vol. 31, no. 1, pp. 147–153, Feb. 2006.
[10] S. Bernstein, *Collected Works*. Moscow: USSR Academy of Sciences, 1952, vol. 1.
[11] S. Bernstein, *Collected Works*. Moscow: USSR Academy of Sciences, 1954, vol. 2.
[12] J. Berchtold and A. Bowyer, "Robust arithmetic for multivariate Bernstein-form polynomials," *Computer-Aided Design*, vol. 32, pp. 681–689, 2000.
[13] R. Farouki and V. Rajan, "On the numerical condition of polynomials in Bernstein form," *Computer-Aided Geometric Design*, vol. 4, no. 3, pp. 191–216, 1987.
[14] J. Garloff, J. Vehi and M. A. Sainz, Eds., "Application of bernstein expansion to the solution of control problems," in *Proc. Workshop on Applications of Interval Analysis to Systems and Control (MISC'99)*, Girona (Spain), 1999, pp. 421–430.
[15] J. Garloff and B. Graf, *The Use of Symbolic Methods in Control System Analysis and Design*. London, UK: Institution of Electrical Engineers (IEE), 1999, ch. Solving Strict Polynomial Inequalities by Bernstein Expansion, pp. 339–352.
[16] R. Martin, H. Shou, I. Voiculescu, A. Bowyer, and G. Wang, "Comparison of interval methods for plotting algebraic curves," *Computer Aided Geometric Design*, vol. 19, pp. 553–587, 2002.
[17] S. Pop, A. Cohen, C. Bastoul, S. Girbal, P. Jouvelot, G.-A. Silber, and N. Vasilache, "Graphite: Loop optimizations based on the polyhedral model for GCC," presented at the 4th GCC Developer's Summit, Ottawa, Canada, 2006.
[18] E. Schweitz, R. Lethin, A. Leung, and B. Meister, "R-stream: A parametric high level compiler," in *10th Annu. Workshop on High Performance Embedded Computing (HPEC 2006)*, J. Kepner, Ed., Lexington, MA, 2006.
[19] S. P. Amarasinghe, J. M. Anderson, M. S. Lam, and C. W. Tseng, "The suif compiler for scalable parallel machines," in *Proc. 7th SIAM Conf. Parallel Processing for Scientific Computing*, 1995.
[20] S. Verdoolaege, R. Seghir, K. Beyls, V. Loechner, and M. Bruynooghe, "Counting integer points in parametric polytopes using Barvinok's rational functions," *Algorithmica*, vol. 48, no. 1, pp. 37–66, Jun. 2007.
[21] S. Verdoolaege, K. Beyls, M. Bruynooghe, and F. Catthoor, "Experiences with enumeration of integer projections of parametric polytopes," in *Proc. 14th Int. Conf. Compiler Construction*, R. Bodík, Ed., Berlin/Heidelberg, 2005, vol. LNCS3443, pp. 91–105.
[22] R. Seghir and V. Loechner, "Memory optimization by counting points in integer transformations of parametric polytopes," in *Proc. Int. Conf. Compilers, Architectures, and Synthesis for Embedded Systems (CASES 2006)*, Seoul, Korea, Oct. 2006.
[23] P. Clauss and V. Loechner, "Parametric analysis of polyhedral iteration spaces," *J. VLSI Signal Process.*, vol. 19, 1998.
[24] B. Meister, "Stating and manipulating periodicity in the polytope model. Applications to program analysis and optimization," Ph.D. dissertation, ICPS, Université Louis Pasteur de Strasbourg, Strasbourg, France, 2004.
[25] V. Loechner, Polylib: A library for manipulating parameterized polyhedra ICPS, Université Louis Pasteur de Strasbourg, France, Tech. Rep., Mar. 1999.
[26] T. Rabl, "Volume calculation and estimation of parameterized integer polytopes," Master's thesis, Universität Passau, Germany, 2006.
[27] H. Ratschek and J. Rokne, *Computer Methods for the Range of Functions*. Chicester, UK: Ellis Horwood, 1984.
[28] G. Farin, *Curves and Surfaces in Computer Aided Geometric Design*. San Diego, CA: Academic Press, 1993.
[29] M. Zettler and J. Garloff, "Robustness analysis of polynomials with polynomial parameter dependency using Bernstein expansion," *IEEE Trans. Autom. Contr.*, vol. 43, pp. 425–431, 1998.
[30] G. M. Ziegler, *Lectures on Polytopes*. Berlin, Germany: Springer-Verlag, 1995.
[31] C. Bauer, A. Frink, and R. Kreckel, "Introduction to the GiNaC framework for symbolic computation within the C++ programming language," *J. Symb. Comput.*, vol. 33, no. 1, pp. 1–12, 2002.
[32] B. Haible, CLN: Class library for numbers. 2006 [Online]. Available: http://www.ginac.de/CLN/
[33] S. Verdoolaege, Barvinok, A library for counting the number of integer points in parametrized and non-parametrized polytopes. 2007 [Online]. Available: http://freshmeat.net/projects/barvinok
[34] P. Feautrier, "Dataflow analysis of array and scalar references," *Int. J. Parallel Programm.*, vol. 20, no. 1, pp. 23–53, 1991.
[35] D. Barthou, J.-F. Collard, and P. Feautrier, "Fuzzy array dataflow analysis," *J. Parallel Distrib. Comput*, vol. 40, no. 2, pp. 210–226, 1997.
[36] H. Zhu, "Computation of memory requirements for multi-dimensional signal processing applications," Ph.D. dissertation, Univ. Illinois, Chicago, 2006.
[37] C. Bastoul, "Code generation in the polyhedral model is easier than you think," in *Proc. 13th Int. Conf. Parallel Architectures and Compilation Techniques (PACT '04)*, Washington, DC, 2004, pp. 7–16.
[38] S. Verdoolaege, H. Nikolov, and T. Stefanov, "PN: A tool for improved derivation of process networks," *EURASIP, J. Embedded Systems, Special Issue on Embedded Digital Signal Processing Systems*, 2007.
[39] P. Clauss, F. J. Fernández, D. Garbervetsky, and S. Verdoolaege, "Symbolic polynomial maximization over convex sets and its application to memory requirement estimation," Université Louis Pasteur, France, ICPS Research Report 06-04, 2006.
[40] K. Beyls and E. D'Hollander, "Generating cache hints for improved program efficiency," *J. Syst. Arch.*, vol. 51, no. 4, pp. 223–250, 2005.
[41] S. Verdoolaege and M. Bruynooghe, M. Beck and T. Stoll, Eds., "Algorithms for weighted counting over parametric polytopes: A survey and a practical comparison," in *2008 Int. Conf. Information Theory and Statistical Learning*, Jul. 2008.
[42] V. Braberman, D. Garbervetsky, and S. Yovine, "A static analysis for synthesizing parametric specifications of dynamic memory consumption," *J. Object Technol.*, vol. 5, no. 5, pp. 31–58, Jun. 2006.
[43] S. Cherem and R. Rugina, "Region analysis and transformation for java programs," in *Proc. 4th Int. Symp. Memory Management (ISMM'04)*, New York, NY, 2004, pp. 85–96.
[44] A. Salcianu and M. Rinard, "Pointer and escape analysis for multithreaded programs," in *Proc. 8th ACM SIGPLAN Symp. Principles and Practices of Parallel Programming (PPoPP'01)*, 2001, pp. 12–23.
[45] V. Braberman, F. Fernández, D. Garbervetsky, and S. Yovine, "Parametric prediction of heap memory requirements," in *Proc. ACM Int. Symp. Memory Management*, Jun. 2008, pp. 141–150.
[46] V. Maslov and W. Pugh, "Simplifying polynomial constraints over integers to make dependence analysis more precise," in *CONPAR 94—VAPP VI, Int. Conf. Parallel and Vector Processing*, Sep. 1994.
[47] W. Blume and R. Eigenmann, "Symbolic range propagation," Univ of Illinois at Urbana-Champaign, Ctr. for Supercomputing Res. & Dev., Tech. Rep. 1381, 1994.
[48] R. A. V. Engelen, K. Gallivan, and B. Walsh, "Parametric timing estimation with the Newton-Gregory formulae," *J. Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1434–1464, Sep. 2006.
[49] A. Darte, R. Schreiber, and G. Villard, "Lattice-based memory allocation," *IEEE Trans. Comput.*, vol. 54, pp. 1242–1257, 2005.
[50] M. Hofmann and S. Jost, "Static prediction of heap usage for first-order functional programs," in *Proc. 30th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL'03)*, Jan. 2003, pp. 185–197.
[51] J. Hughes and L. Pareto, "Recursion and dynamic data-structures in bounded space: towards embedded ml programming," in *Proc. ICFP '99*, 1999, pp. 70–81.
[52] L. Unnikrishnan, S. Stoller, and Y. Liu, "Optimized live heap bound analysis," in *Proc. VMCAI '03*, Jan. 2003, vol. 2575, pp. 70–85.
[53] W.-N. Chin, H. H. Nguyen, S. Qin, and M. C. Rinard, C. Hankin and I. Siveroni, Eds., "Memory usage verification for oo programs," in *Proc. 12th Int. Symp. Static Analysis (SAS 2005)*, 2005, vol. 3672, pp. 70–86.

Philippe Clauss received the Ph.D. degree in computer science in 1990.

He is a full-time Professor at the CNRS Laboratory LSIIT of the University of Strasbourg, Strasbourg, France. His research interests include parallelizing and optimizing compilers, software static and dynamic analysis and optimizations.

Federico Javier Fernández received the M.Sc. degree from the University of Buenos Aires, Argentina. While doing his master's thesis on static analysis, he worked with the Bernstein basis and its applications to memory counting techniques. He is now working towards the Ph.D. degree, also at the University of Buenos Aires, while working on some unrelated projects dealing with street transit analysis and optimization.

Diego Garbervetsky received the Ph.D. degree in computer science.

He is now a full-time Associate Professor in the Computer Science Department of the University of Buenos Aires, Argentina. His research field is embedded systems, formal verification and program analysis focused on prediction of dynamic memory utilization.

Sven Verdoolaege received the Masters degrees in computer science and artificial intelligence as well as the Ph.D. degree in computer science from the Katholieke Universiteit Leuven, Belgium, in 1998, 1999, and 2005, respectively.

From 2005 until 2007, he was a Postdoc at the Leiden Institute of Advanced Computer Science in The Netherlands. He is currently a Postdoc at the Katholieke Universiteit Leuven, Belgium. His research interests include loop transformations, integer points in polyhedra, and equivalence verification.