Symbolic Polytopes for Quantitative Interpolation and Verification

Klaus v. Gleissenthall[1], Boris Köpf[2], and Andrey Rybalchenko[3]

[1] TUM [2] IMDEA [3] MSR

Abstract. Proving quantitative properties of programs, such as bounds on resource usage or information leakage, often leads to verification conditions that involve cardinalities of sets. Existing approaches for dealing with such verification conditions operate by checking cardinality bounds for given formulas. However, they cannot synthesize formulas that satisfy given cardinality constraints, which limits their applicability for inferring cardinality-based inductive arguments.

In this paper we present an algorithm for synthesizing formulas for given cardinality constraints, which relies on the theory of counting integral points in symbolic polytopes. We cast our algorithm in terms of a cardinality-constrained interpolation procedure, which we put to work in a solver for recursive Horn clauses with cardinality constraints based on abstraction refinement. We implement our technique and describe its evaluation on a number of representative examples.

1 Introduction

Proving quantitative properties of programs often leads to verification conditions that involve cardinalities of sets and relations over program states. For example, determining the memory requirements for memoization reduces to bounding the cardinality of the set of argument values passed to a function, and bounding information leaks of a program reduces to bounding the cardinality of the set of observations an attacker can make.

A number of recent advances for discharging verification conditions with cardinalities consider extensions of logical theories with cardinality constraints, such as set algebra and its generalizations [23, 29, 30], linear arithmetic [13, 37], constraints over strings [25], as well as general SMT based settings [15]. At their core, these approaches operate by *checking* whether a cardinality bound holds for a given formula that describes a set of values. However, they cannot *synthesize* formulas that satisfy given cardinality constraints. As a consequence, the problem of automatically inferring cardinality-based inductive arguments that imply a specified assertion remains an open challenge.

In this paper, we present an approach for synthesizing linear arithmetic formulas that satisfy given cardinality constraints. Our approach relies on the theory of counting integral points in polytopes, however, instead of computing the cardinality of a given polytope (the typical use case of this theory) our approach synthesizes a polytope for a given cardinality constraint. Our synthesizer internally organizes the search space in terms of *symbolic* polytopes. Such polytopes are represented using symbolic vertices and hyperplanes, together with certain well-formedness constraints. We derive an expression for the number of points in the polytope in terms of this symbolic representation, which leads to a set of constraints that at the same time represent the shape *and* the cardinality of the polytope. For this, we restrict our attention to the class of *unimodular* polytopes. Unimodularity can be concisely described using constraints and provides an effective means for reducing the search space while being sufficiently expressive. We then resort to efficient SMT solvers specifically tuned to deal with the resulting kind of non-linear constraints, e.g., Z3 [14]. We cast our approach in terms of an algorithm $\#\text{ITP}_{\text{LIA}}$ for cardinality constrained interpolation, that is, $\#\text{ITP}_{\text{LIA}}$ generates formulas that satisfy cardinality constraints along with implication constraints.

We put cardinality-constrained interpolation to work within an automatic verification method #HORN for inferring cardinality-based inductive program properties, based on abstraction and its counterexample-guided refinement. Specifically, #HORN is a solver for recursive Horn clauses with cardinality constraints. We rely on Horn clauses as basis because they serve as a language for describing verification conditions for a wide range of programs, including those with procedures and multiple threads [7, 17, 32]. Adding recursion enables representing verification conditions that rely on inductive reasoning, such as loop invariants or procedure summaries. By offering support for cardinalities directly in the language in which we express verification conditions, our solver can effectively leverage the interplay between the qualitative and quantitative (cardinality) aspects of the constraints to be solved.

We implemented $\#\text{ITP}_{\text{LIA}}$ and #HORN and applied them to analyze a collection of examples that show

- how a variety of cardinality-based properties (namely, bounds on information leaks, memory usage, and execution time) and different program classes (namely, while programs and programs with procedures) can be expressed and analyzed in a uniform manner.
- that our approach can establish resource bounds on examples from the recent literature at competitive performance and precision, and that it can handle examples whose precise analysis is out of scope of existing approaches.
- that our approach can be used for synthesizing a padding-based countermeasure against timing side channels, for a given bound on tolerable leakage.

In summary, our paper contributes and puts to work a synthesis method for polytopes that satisfy cardinality constraints, based on symbolic integer point counting algorithms.

2 Example

We consider a procedure `mcm` for *Matrix chain multiplication* [12] that recursively computes the cost of multiplying matrices $M_0, \ldots, M_n$ with optimal bracketing.

mcm calls c(k) to obtain the cost of multiplying matrices M_k and M_{k+1}. Executing mcm (i, j) computes the minimal cost of multiplying sequence $M_i, \ldots, M_j$. Even though the number of recursive function calls is exponential in n, mcm can be turned into an efficient algorithm by applying memoization. The amount of memory required to store results of recursive calls is bounded by $\frac{(n+1)\cdot(n+2)}{2}$, as mcm is only called with ordered pairs of arguments.

```
int mcm(int i, int j) {
 if (i == j) return 0;
 int minCost = infty;
 for (int k=i; k <= j-1; k++) {
  int v = mcm(i, k)+mcm(k+1, j)+c(k);
  if (v < minCost) minCost = v;
 }
 return minCost;
}
int main(n){
  mcm(0, n);
}
```

Proving such a bound requires reasoning about recursive procedure calls as well as tracking dependencies between variables i and j, i.e. estimating the range of each variable in isolation and combining the estimates is not precise enough.

When using #Horn, we first set up recursive Horn constraints on an assertion $args(i, j, n)$ that contains all triples (i, j, n) such that calling main(n) leads to a recursive call mcm(i, j), following [17]. Then, #Horn solves these constraints using a counterexample guided abstraction and refinement based procedure. As an intermediate step, #Horn deals with an interpolation query that requires finding a polytope φ_{args} over i, j and n such that

$$n \geq 2 \land (i = 0 \land j = n \lor i = 1 \land j = 1) \rightarrow \varphi_{args} \qquad (1)$$

$$n \geq 0 \rightarrow |\{(i, j) \mid \varphi_{args}\}| \leq \tfrac{(n+1)\cdot(n+2)}{2} \; . \qquad (2)$$

Constraint (1) requires that φ_{args} contains triples obtained by symbolically executing mcm, a typical interpolation query, while (2) ensures that φ_{args} satisfies the bound by referring to the cardinality of φ_{args} through an application of cardinality operator $|\cdot|$.

Given (1) and (2), #Itp_{LIA} computes the solution $\varphi_{args} = (0 \leq i \leq 1 \land i \leq j \leq n \land n \geq 2)$. The cardinality of $\{(i, j) \mid \varphi_{args}\}$ is $2n + 1$, hence φ_{args} satisfies the above bound. #Horn uses this solution to refine the abstraction function. In particular, it starts using the predicate $i \leq j$, which is crucial for tracking that mcm is only called on ordered pairs.

3 Counting integer points in polytopes

In this section, we first revisit the theory of counting integral points in polytopes. We then discuss the derivation of expressions for the number of integer points in unimodular polytopes with symbolic vertices and hyperplanes.

Preliminaries Let $g_1, \ldots, g_d$ be d-dimensional vectors. A *cone* with generators $g_1, \ldots, g_d$ is the set of all positive linear combinations of its generators. A cone is *unimodular* if the absolute value of the determinant of the matrix

$(g_1 \ \dots \ g_d)$ is equal to one. The *vertex cone* of a polytope P at vertex v is the smallest cone that originates from v and that includes P. We let $g_{v1}, \dots, g_{vd}$ denote its generators. Finally, a polytope P is unimodular if all its vertex cones are unimodular.[1]

Generating functions The integral points contained in a set $S \subseteq \mathbb{R}^d$ in $d-$dimensional space can be represented in terms of a *generating function* $f(S, x)$ which is a sum of monomials, one per integer point in S, defined as follows

$$f(S, x) = \sum_{m \in S \cap \mathbb{Z}^d} x^m, \tag{1}$$

where for $m = (m_1, \dots, m_d)$ we define $x^m = x_1^{m_1} \cdot \dots \cdot x_d^{m_d}$. This generating function is a Laurent series, i.e. its terms may have negative degree. Note that, for finite S, the value of $f(S, x)$ at $x = (1, \dots, 1)$, corresponds to the number of integer points in S.

Rational function representation Generating functions are a powerful tool for counting integer points in polytopes. This is due to two key results: First, Brion's theorem [9] allows to decompose the generating function of a polytope into the sum of the generating functions of its vertex cones. Second, the generating function of *unimodular* vertex cones can be represented through an equivalent yet short rational function. This rational function representation relies on a generalization of the equivalence $\frac{1}{1-x} = (1 + x + x^2 + x^3 + \dots)$, which provides a concise representation of the set $\{0, 1, 2, 3, \dots\}$.

Overall one obtains the following rational function representation for a unimodular polytope P with vertices $\mathcal{V}$:

$$r(P, x) = \sum_{v \in \mathcal{V}} \frac{x^v}{(1 - x^{g_{v1}}) \cdots (1 - x^{g_{vd}})} \tag{2}$$

Here, each summand represents the generating function of the vertex cone at v with generators $g_{v1}, \dots, g_{vd}$. Rational function representations for arbitrary polytopes can be obtained through Barvinok's algorithm [3] that decomposes arbitrary vertex cones into unimodular cones.

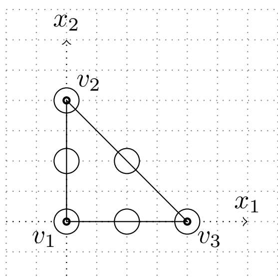
Generating function evaluation Since $x = (1, \dots, 1)$ is a singularity of $r(P, x)$ we cannot directly obtain the number of points in the polytope by evaluation, as such evaluation would lead to division by zero. However, this singularity can be eliminated by applying the Laurent series expansion of $r(P, x)$ around $x = (1, \dots, 1)$. The expansion requires first a reduction of $r(P, x)$ from a multivariate polynomial over $(x_1, \dots, x_d)$ to a univariate polynomial over y, see [13]. Let G denote the set of generators for a given polytope. The reduction from x to y is based on a vector $\mu = (\mu_1, \dots, \mu_d)$ such that

[1] Appendix A provides examples and alternative equivalent defintions of unimodularity. For more details, see e.g. [3, 4, 13].

$$\bigwedge_{g \in G} \mu \cdot g \neq 0 \ . \tag{3}$$

We replace x_i by y^{μ_i}, for each $i \in 1..d$. Equation (3) ensures that no factor in the denominator of Equation (2) becomes 0, and hence avoids introduction of singularities. Let $sub(r(P,x),y)$ denote the result of the above substitution. Then, the constant term of the Laurent expansion of $sub(r(P,x),y)$ around $y=1$ yields the desired count. Computing Laurent series expansions is a standard procedure, see e.g. Wolfram Alpha [38].

Example 1. Consider the unimodular polytope $P = (x_1 \geq 0 \wedge x_2 \geq 0 \wedge x_1 + x_2 \leq 2)$ of dimension $d = 2$. P is given by the vertices $v_1 = (0\ 0)$, $v_2 = (0\ 2)$, and $v_3 = (2\ 0)$. P contains 6 integer points, as shown by the circles below.



The generators of the vertex cones are given by

$$\begin{aligned} g_{v_1 1} &= (0\ 1) & g_{v_1 2} &= (1\ 0) \\ g_{v_2 1} &= (0\ -1) & g_{v_2 2} &= (1\ -1) \\ g_{v_3 1} &= (-1\ 0) & g_{v_3 2} &= (-1\ 1). \end{aligned}$$

Equation (2) yields the following rational generating function $r(P,x)$.

$$\frac{x_1^0 x_2^0}{(1-x_1^0 x_2^1)(1-x_1^1 x_2^0)} + \frac{x_1^0 x_2^2}{(1-x_1^0 x_2^{-1})(1-x_1^1 x_2^{-1})} + \frac{x_1^2 x_2^0}{(1-x_1^{-1} x_2^0)(1-x_1^{-1} x_2^1)}$$

Applying the substitution with $\mu = (-1\ 1)$ yields the expression $sub(r(P,x),y)$.

$$\frac{1}{(1-y)(1-y^{-1})} + \frac{y^2}{(1-y^{-1})(1-y^{-2})} + \frac{y^{-2}}{(1-y)(1-y^2)}$$

Computing the series expansion using the Wolfram alpha command *series sub*$(r(P,x),y)$ *at* $y = 1$ produces $\ldots 5(y-1)^3 + 5(y-1)^2 + 6$, with the constant coefficient 6 yielding the expected count. ■

Symbolic cardinality expression The rational function representation of the generating function of a unimodular polytope shown in Equation 2 refers to the polytope's vertices and to the generators of its vertex cones. However, these generators and vertices do not have to be instantiated to concrete values in order for the evaluation of the generating function to be possible [37]. That is, the evaluation of the generating function can be carried out *symbolically* yielding a formula that expresses the cardinality of a polytope as a function of its generators, vertices, and a vector μ.

In our algorithm, we will use SymConeCard(v, G, μ) to refer to the result of the symbolic evaluation of the generating function for the cone of a symbolic vertex v, with generators G. By summing up $\textsc{SymConeCard}(v, G, \mu)$ for all vertex cones we obtain a symbolic expression of the number of integer points in a symbolic polytope.[2]

Example 2. The cardinality of a two dimensional polytope with symbolic vertices v_1, v_2, v_3 and generators g_{v_i1} and g_{v_i2}, with $i \in 1..3$, is given by $\sum_{i=1}^{3} \textsc{SymConeCard}(v_i, \{g_{v_i1}, g_{v_i2}\}, \mu)$, where

$$\begin{aligned}&\textsc{SymConeCard}(v_i, \{g_{v_i1}, g_{v_i2}\}, \mu)\\&= (\mu_1^2 + 3\mu_1(\mu_2 - 2\mu_v - 1) + \mu_2^2 - 3\mu_2(2\mu_v + 1) + 6\mu_v^2 + 6\mu_v + 1)(12\mu_1\mu_2)^{-1}\\&\quad \text{with } \mu_1 = \mu \cdot g_{v_i1},\ \mu_2 = \mu \cdot g_{v_i2} \text{ and } \mu_v = \mu \cdot v_i.\end{aligned}$$

■

Note that, in order for $\textsc{SymConeCard}(v, G, \mu)$ to yield a valid count, the vertices and generators must satisfy a number of conditions, e.g., the symbolic cones need to be unimodular and the employed vector μ needs to satisfy Equation (3). We next present our interpolation procedure $\#\textsc{Itp}_{\text{LIA}}$ that creates constraints for ensuring these conditions.

4 Interpolation with cardinality constraints

In this section, we first define interpolation with cardinality constraints. Then we present the interpolation procedure $\#\textsc{Itp}_{\text{LIA}}$ that generates constraints on the cardinality of an interpolant and solves them using an SMT solver.

Cardinality interpolation Let k be a variable and let w be a tuple of variables. Let φ and ψ be constraints in a given first-order theory. Then, a *cardinality constraint* is an expression of the form

$$|\{w \mid \varphi\}| = k \wedge \psi$$

where $|\cdot|$ denotes the set cardinality operator. We call the free variables of φ that do not occur in w *parameters*. A cardinality constraint is *parametric* if it has at least one parameter and *non-parametric* otherwise. Parameters define free variables of the expression ψ that constrains the cardinality.

Example 3. Consider the theory of linear integer arithmetic. The cardinality constraint $|\{x \mid 0 \leq x \leq 10\}| = k \wedge k \leq 20$ is non-parametric, whereas the constraint $|\{x \mid 0 \leq x \leq n\}| = k \wedge k \leq n+1$ is parametric in n. Both constraints are valid, since $|\{x \mid 0 \leq x \leq 10\}| = 11$ and $|\{x \mid 0 \leq x \leq n\}| = n + 1$. ■

[2] This step relies on the fact that evaluating the generating function for each vertex cone separately and summing the results is equivalent to evaluating the sum of generating functions.

```
function #Itp_LIA(w, φ⁻, φ⁺, ψ, Tmpl)
  Cons := true
  SymCard := 0
  d := length of w
  μ := vector of d fresh variables
  H_V := ⋃{Tmpl(v) | v ∈ V}
  for each v ∈ V do
    H := Tmpl(v)
    G := ∅
    for each H ∈ H do
      g_vH := vector of d fresh variables
      G := {g_vH} ∪ G
    Cons := Cons ∧ Vert(v, H, H_V) ∧ Genr(v, H, G, μ) ∧ Unim(v, G)
    SymCard := SymCard + SymConeCard(v, G, μ)
  Cons := Cons ∧ Impl(φ⁻, ⋀H_V) ∧ Impl(⋀H_V, φ⁺)
  return SMTSolve(Cons ∧ Impl(SymCard = k, ψ(k)))
```

Fig. 1: $\#\textsc{Itp}_{\text{LIA}}$ for cardinality constrained interpolantion for given $\textsc{Tmpl}$.

Assume constraints φ^- and φ^+ such that φ^- implies φ^+. A *cardinality constrained interpolant* for φ^-, φ^+, and cardinality constraint $|\{w \mid \varphi\}| = k \land \psi$ is a constraint φ such that 1) φ^- implies φ, 2) φ implies φ^+, and 3) $|\{w \mid \varphi\}| = k \land \psi$ is valid. For a parametric cardinality constraint, we say that the interpolation problem is parametric, and call it non-parametric otherwise.

Example 4. Let $\varphi^- = (x = 0 \land n \geq 0)$ and $\varphi^+ = \mathit{true}$. Then $\varphi = (0 \leq x \leq n)$ is an interpolant that satisfies the cardinality constraint $|\{x \mid \varphi\}| = k \land k \leq n+1$. For $\varphi^- = \mathit{false}$, $\varphi^+ = x \geq 0$ and cardinality constraint $|\{x \mid \varphi\}| = k \land 1 \leq k \leq 10$ the constraint $\varphi = (0 \leq x \leq 5)$ is a cardinality constrained interpolant. ■

Note that our definition of interpolation differs from the standard, cardinality-free definition given e.g. in [27] in that we do not require the free variables in φ to be common to both φ^- and φ^+. We exclude this requirement because it appears to be overly restrictive for the cardinality setting, as the cardinality constraint imposes a lower/upper bound in addition to φ^- and φ^+. In particular, the common variables condition rules out both interpolants in Example 4, as the set of common variables is empty in both cases.

In this paper, we focus on cardinality constraints with φ in linear arithmetic and ψ in (non-)linear arithmetic, which is an important combination for applications in software verification.

Interpolation algorithm We present an algorithm $\#\textsc{Itp}_{\text{LIA}}$ for interpolation with cardinality constraints. For simplicity of exposition, we first consider the non-parametric case and discuss the parametric case in Section 5.

$\#\textsc{Itp}_{\text{LIA}}$ finds an interpolant φ in a space of polytope candidates that is defined by a template. We rely on a function $\textsc{Tmpl}$ that maps a symbolic vertex $v \in \mathcal{V}$ to a set of symbolic hyperplanes that are determined to intersect in v, thereby partially determining the shape of φ. Each hyperplane $H \in \textsc{Tmpl}(v)$ is of the form $c_H \cdot w = \gamma_H$.

The algorithm $\#\textsc{Itp}_{\text{LIA}}$ is described in Figure 1. We collect a constraint $\textsc{Cons}$ over the symbolic vertices and symbolic hyperplanes of φ, which ensures that any solution yields a unimodular polytope that satisfies conditions 1) – 3) of the definition of cardinality interpolation. In particular, $\#\textsc{Itp}_{\text{LIA}}$ ensures that the cardinality of φ satisfies ψ by constructing a symbolic expression $\textsc{SymCard}$ on the cardinality of φ in line 13, and requiring that this expression satisfies the cardinality constraint ψ in line 15. Lines 12 produces well-formedness constraints $\textsc{Vert}(v, \mathcal{H}, \mathcal{H}_\mathcal{V})$ and $\textsc{Genr}(v, \mathcal{H}, G)$ that ensure a geometrically well-formed instantiation of the template $\textsc{Tmpl}$. The final conjunct in line 12 poses constraints on the generators of the vertex cones in φ that ensure their unimodularity, as required by Section 3. Finally, line 14 produces constraints that ensure the validity of the implications $\varphi^- \to \varphi$ and $\varphi \to \varphi^+$. The resulting constraint $\textsc{Cons}$ is passed to an SMT solver that either returns a valuation of symbolic vertices and hyperplanes and hence determines φ, or fails.

Constraint generation We will now describe the constraint generation of $\#\textsc{Itp}_{\text{LIA}}$ in more detail. For each symbolic vertex v we make sure that it lies on the hyperplanes determined by $\textsc{Tmpl}(v)$ and in the appropriate half-space wrt. the remaning hyperplanes. This is achieved by the following constraint.

$$\textsc{Vert}(v, \mathcal{H}, \mathcal{H}_\mathcal{V}) = \bigwedge_{H \in \mathcal{H}} c_H \cdot v = \gamma_H \;\; \wedge \bigwedge_{H \in \mathcal{H}_\mathcal{V} \setminus \mathcal{H}} c_H \cdot v < \gamma_H$$

By making certain inequalities strict, we ensure that the polytope does not collapse into a single point, since in this case Brion's theorem does not hold.

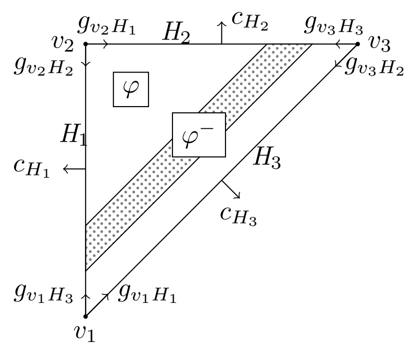
$\textsc{SymConeCard}$ and $\textsc{Unim}$ refer to the generators of vertex cones determined by $\textsc{Tmpl}$. Hence we produce a constraint that defines these generators in terms of symbolic hyperplanes. Let g_{vH} denote the generator of the cone at vertex v that lies in the half-space described by hyperplane H. Then we constrain the generators of the cone at v as follows.

$$\begin{aligned}\textsc{Genr}(v, \mathcal{H}, G, \mu) = \textstyle\bigwedge_{H \in \mathcal{H}} (c_H \cdot g_{vH} \leq 0 \wedge \mu \cdot g_{vH} \neq 0 \\ \wedge \textstyle\bigwedge_{H' \in \mathcal{H} \setminus \{H\}} c_{H'} \cdot g_{vH} = 0)\end{aligned}$$

Here we require each generator g_{vH} to lie on the facet formed by the intersection of all hyperplanes $H' \in \mathcal{H} \setminus \{H\}$, and pointing in the appropriate half-space wrt. H. Additionally the generator is constrained according to Equation 3. With the generators defined, we can ensure the unimodularity of vertex cones of the polytope by $\textsc{Unim}(v, G) = (abs(det(g_{vH_1}, \ldots, g_{vH_d})) = 1)$, where $G = \{g_{vH_1}, \ldots, g_{vH_d}\}$. We then use $\textsc{SymConeCard}(v, G, \mu)$ to denote the counting expression of the symbolic cone of vertex v for our generators. We construct the counting expressions for the entire symbolic polytope φ by taking the sum over counting expressions for its vertex cones.

Finally, we generate constraints Impl for the implication conditions $\varphi^- \to \varphi$ and $\varphi \to \varphi^+$ by applying Farkas' lemma [33], which is a standard tool for such tasks [11, 31]. This lemma states that every linear consequence of a satisfiable set of linear inequalities can be obtained as a non-negative linear combination of these inequalities. Formally, if $Aw \le b$ is satisfiable and $Aw \le b$ implies $cw \le \gamma$ then there exists $\lambda \ge 0$ such that $\lambda A = c$ and $\lambda b \le \gamma$. When dealing with integers, Farkas' lemma is sound but not complete, see the following discussion on completeness. Our implementation of Impl handles non-conjunctive implication constraints by a standard method based on DNF conversion and Farkas' lemma.

Example 5. Consider $\varphi^- = (1 \le x \wedge x{-}y \le 1 \wedge x{-}y \ge -1 \wedge y \le z \wedge z \le 10)$, $\varphi^+ = \mathit{true}$, $w = (x, y)$, and $\psi = (k \le 120)$. The solution φ is a polytope formed by three vertices $\mathcal{V} = \{v_1, v_2, v_3\}$. It is bounded by the supporting hyperplanes $\mathcal{H}_\mathcal{V} = \{H_1, H_2, H_3\}$ with normal vectors c_{H_1}, c_{H_2} and c_{H_3}, respectively. In our example, we use Tmpl such that $v_1 \mapsto \{H_1, H_3\}$, $v_2 \mapsto \{H_1, H_2\}$, and $v_3 \mapsto \{H_2, H_3\}$, restricting φ to a triangular shape.



We obtain the following constraints.

$$\text{Vert}(v_1, \{H_1, H_3\}, \mathcal{H}_\mathcal{V}) = (c_{H_1} \cdot v_1 = \gamma_{H_1} \wedge c_{H_3} \cdot v_1 = \gamma_{H_3} \wedge c_{H_2} \cdot v_1 < \gamma_{H_2}) \ ,$$
$$\text{Vert}(v_2, \{H_1, H_2\}, \mathcal{H}_\mathcal{V}) = (c_{H_1} \cdot v_2 = \gamma_{H_1} \wedge c_{H_2} \cdot v_2 = \gamma_{H_2} \wedge c_{H_3} \cdot v_2 < \gamma_{H_3}) \ ,$$
$$\text{Vert}(v_3, \{H_2, H_3\}, \mathcal{H}_\mathcal{V}) = (c_{H_2} \cdot v_3 = \gamma_{H_2} \wedge c_{H_3} \cdot v_3 = \gamma_{H_3} \wedge c_{H_1} \cdot v_3 < \gamma_{H_1}) \ .$$

We get the following constraints on generators.

$$\text{Genr}(v_1, \{H_1, H_3\}, \{g_{v_1H_1}, g_{v_1H_3}\}, \mu) \ =$$
$$(c_{H_1} \cdot g_{v_1H_1} \le 0 \wedge c_{H_3} \cdot g_{v_1H_1} = 0 \wedge c_{H_3} \cdot g_{v_1H_3} \le 0 \wedge c_{H_1} \cdot g_{v_1H_3} = 0)$$
$$\text{Genr}(v_2, \{H_1, H_2\}, \{g_{v_2H_1}, g_{v_2H_2}\}, \mu) \ =$$
$$(c_{H_1} \cdot g_{v_2H_1} \le 0 \wedge c_{H_2} \cdot g_{v_2H_1} = 0 \wedge c_{H_2} \cdot g_{v_2H_2} \le 0 \wedge c_{H_1} \cdot g_{v_2H_2} = 0)$$
$$\text{Genr}(v_3, \{H_2, H_3\}, \{g_{v_3H_2}, g_{v_3H_3}\}, \mu) \ =$$
$$(c_{H_2} \cdot g_{v_3H_2} \le 0 \wedge c_{H_3} \cdot g_{v_3H_2} = 0 \wedge c_{H_3} \cdot g_{v_3H_3} \le 0 \wedge c_{H_2} \cdot g_{v_3H_3} = 0)$$

and unimodularity restrictions:

$$\mathit{abs}(\mathit{det}(g_{v_1H_1}, g_{v_1H_3})) = \mathit{abs}(\mathit{det}(g_{v_2H_1}, g_{v_2H_2})) = \mathit{abs}(\mathit{det}(g_{v_3H_2}, g_{v_3H_3})) = 1.$$

The implication constraints in matrix notation are

$$\overbrace{\begin{pmatrix}-1 & 0\\ 1 & -1\\ -1 & 1\\ 0 & 1\end{pmatrix}}^{A}\begin{pmatrix}x\\ y\end{pmatrix} \leq \overbrace{\begin{pmatrix}-1\\ 1\\ 1\\ 10\end{pmatrix}}^{b} \quad \rightarrow \quad \overbrace{\begin{pmatrix}c_{11} & c_{12}\\ c_{21} & c_{22}\\ c_{31} & c_{32}\end{pmatrix}}^{C}\begin{pmatrix}x\\ y\end{pmatrix} \leq \overbrace{\begin{pmatrix}\gamma_1\\ \gamma_2\\ \gamma_3\end{pmatrix}}^{\gamma}$$

where, for each $i \in \{1, 2, 3\}$, we obtain the following constraints for H_i by an application of Farkas' lemma $\exists\lambda^i : \lambda^i \geq 0 \wedge \lambda^i A = C_i \wedge \lambda^i b \leq \gamma_1$. We pass the constraints to an SMT solver and obtain the solution $\varphi = (1 \leq x \ \wedge \ y \leq 10 \ \wedge \ y \geq x - 3)$ with $|\{(x, y) \mid \varphi\}| = 91$. ■

Theorem 1 (Soundness). *If* $\#\textsc{Itp}_{\textsc{LIA}}(w, \varphi^-, \varphi^+, \psi, \textsc{Tmpl})$ *returns a solution* φ*, then* φ *is a cardinality constrained interpolant for* φ^- *and* φ^+ *and cardinality constraint* $|\{w \mid \varphi\}| = k \wedge \psi$*.*

Proof. We show that φ satisfies conditions 1) to 3). Conditions 1) and 2) follow from the use of Farkas' lemma. Since the conditions posed by $\textsc{Vert}(v, \mathcal{H}, \mathcal{H}_\mathcal{V})$ ensure that each vertex is active (part of the polytope) and that vertices are distinct, Brion's theorem is applicable and hence the generating function of φ can be expressed as the sum of the generating functions of its vertex cones. Each of φ's vertex cones is unimodular by constraints $\textsc{Unim}(v, G)$ and its generating function is hence given by the expression in Equation 2. Summing over the evaluated rational generating functions of the vertex cones is equivalent to evaluating the sum of the rational generating functions by the fact that Laurent expansion distribute over sums. As a consequence the expression SymCard corresponds to the cardinality of φ and, by the constraint in Line 15 in Figure 1, satisfies the cardinality constraint ψ.

Completeness For a given template, our method returns a solution whenever a solution expressed by the template exists, yet subject to the following two sources of incompleteness. First, solving non-linear integer arithmetic constraints is an undecidable problem and hence the call to SMTSolve may (soundly) fail. Second, Farkas' lemma is incomplete over the integers. Note that these sources of incompleteness did not strike on benchmarks from the literature, see Section 7.

5 Interpolation with parametric cardinalities

We now briefly discuss the parametric interpolation problem by contrasting it with the non-parametric case. Computing the number of integer points in a polytope in terms of a parameter uses the techniques described in Section 3 (see Appendix B for an example). Hence, we can obtain the cardinality of a symbolic polytope in terms of a parameter in a similar fashion. The key challenge we face when extending cardinality constrained interpolation to the parametric case is a quantifier alternation. While in the non-parametric case, the constraints CONS are quantified as $\exists \mathcal{H}_{\mathcal{V}}\ \exists \mathcal{V} : \text{CONS}$, introducing parameters changes the quantifier structure to $\exists \mathcal{H}_{\mathcal{V}}\ \forall p\ \exists \mathcal{V} : \text{CONS}$, where p is a tuple of parameters in the cardinality constraint. The alternation stems from the fact that the parameter valuation detemines the intesection points, that is, the vertices, for parametric polytopes. This alternation has two implication on the computation of interpolants: First, for different values of p the number of vertices of a polytope can vary due to changes in the relative position of the bounding hyperplanes. As a consequence, templates with fixed number of vertices are only valid for a specific parameter range, which is called a *chamber* [37]. We deal with this aspect by considering a predicate *cmb* that restricts the parameter range to the appropriate chamber and that satisfies the implication constraints. We then conjoin *cmb* to the inferred polytope. [3]

Second, solving the cardinality constraint requires quantifier elimination for non-linear arithmetic. For this task we devise a constraint-based method ensuring positivity of a polynomial on a given range by referring to its roots. We provide a short description of this method together with an example of applying our interpolation method on a parametric interpolation query in Appendix C.

6 Verification of programs with cardinality constraints

In this section, we sketch our algorithm #HORN for solving sets of Horn clauses with cardinality constraints. We choose Horn clauses as a basis for representing our verification conditions as they provide a uniform way to encode a variety of verification tasks [5, 6, 8, 17]. The interpolation procedure $\#\text{ITP}_{\text{LIA}}$ presented in Section 4 is a key ingredient for, but not restricted to #HORN.

Horn clauses with cardinality constraints A *Horn clause* is a formula of the form $\varphi_0 \wedge q_1 \wedge \cdots \wedge q_k \rightarrow H$ where φ_0 is a linear arithmetic constraint, and $q_1, \ldots, q_k$ are uninterpreted predicates that we refer to as *queries*. We call the left-hand side of the implication *body* and the right-hand side *head* of the clause. H can either be a constraint φ, a query q, or a cardinality constraint of the form $|\{w \mid q\}| \leq \eta$, where η is a polynomial. By restricting cardinality constraints over queries to this shape, we ensure monotonicity, which is key for the soundness of over-approximation. For a clause $\varphi_0 \wedge q_1 \wedge \cdots \wedge q_k \rightarrow q$, we say that q *depends* on queries $q_1, \ldots, q_k$. We call a set of clauses *recursive* if the dependancy relation contains a cycle and *non-recursive* otherwise. For an example set of Horn clauses with cardinality constraints see Appendix D.

For the semantics, we consider a *solution function* Σ that maps each query symbol q occurring in a given set of clauses into a constraint. The satisfaction relation $\Sigma \models cl$ holds for a clause $cl = (\varphi_0 \wedge q_1 \wedge \cdots \wedge q_k \rightarrow H)$ iff the body of cl entails the head, after replacing each q by $\Sigma(q)$. The lifting from clauses to sets of clauses is canonical.

[3] Note that generators do not depend on the constant terms of the hyperplanes, which is why their constraints are not affected by variations in the parameters.

Algorithm description #Horn takes as input a set C of recursive Horn clauses with cardinality constraints and produces as output either a solution to the clauses or a counterexample. Due the undecidability caused by recursion, #Horn may not terminate. The solver executes the following main steps: abstract inference, property checking, and refinement.

Abstract inference We iteratively build a solution for the set of *inference clauses* $\mathcal{I} = \{cl \in C \mid cl = (\ldots \to q)\}$ by performing logical inference until a fixpoint is reached. This step uses abstraction to ensure that the inference terminates, where the abstraction is determined by a set of predicates $Preds$. This step is standard [17], as clauses $\mathcal{I}$ do not contain cardinality constraints.

Property checking We check whether the constructed solution satisfies all *property clauses* in $\mathcal{P} = C \setminus \mathcal{I}$. The novelty in #Horn is the check for satisfaction of cardinality constraints $|\{w \mid \varphi\}|$, where φ is a linear arithmetic constraint. Here we rely on a parametric extension of Barvinok's algorithm [37], which on input φ returns a set of tuples $\mathscr{B}(\varphi, w) = \{(cmb_1, c_1),\ \ldots\}$ such that whenever the constraint cmb_i holds, the cardinality of $|\{w \mid \varphi\}|$ is given by the expression c_i, which may either be a polynomial c_i or ω (which denotes the unbounded case). We hence reduce checking satisfaction of the cardinality constraint $|\{w \mid \varphi\}|$ to checking the following implication.

$$\models \bigwedge_{(cmb,c)\in\mathscr{B}(\varphi,w)} (cmb \to c \leq \eta)$$

If the check succeeds, the algorithm returns the solution. Otherwise, the algorithm proceeds to a refinement phase in order to analyse the derivation that led to the violation of the property clause.

Refinement We construct a counterexample, i.e., a set CEX of recursion-free Horn clauses with cardinality constraints that represents the derivation that led to the violation of the property clause. This counterexample may either be genuine or spurious due to abstraction. To determine which it is, we rely on a solver for *non-recursive* clauses with cardinality constraints that either produces a solution for the clauses or reports that no such solution exists. If no solution exists, the algorithm returns the counterexample that represents a genuine error derivation. Otherwise it uses #Itp$_{\text{LIA}}$ to eliminate the cardinality constraint from the clauses thus producing a set of *cardinality-free* Horn clauses. We solve these clauses using existing methods [20] and obtain a set of predicates that we use to refine the abstraction.

7 Experiments

We implemented our method in SICStus Prolog, and use its built-in constraint solver for the simplification and projection of linear constraints, HSF [17] for solving recursion- and cardinality-free Horn clauses, and Z3 [14] for non-linear/boolean constraint solving. We use barvinok [35] for checking whether a solution candidate satisfies a cardinality constraint. We use a 1.3 Ghz Intel Core i5 computer with 4 GB of RAM.

Program	**Bound**	**Time**
Dis1 [19]	$max(n - x_0) + max(m - y_0)$	0.19s
Dis2 [19]	$n - x_0 + m - z_0$	0.17s
SimpleSingle [19]	n	0.11s
SequentialSingle [19]	n	0.11s
NestedSingle [19]	$n + 1$	0.15s
SimpleSingle2 [19]	$max(n, m)$	0.13s
SimpleMultiple [19]	$n + m$	0.16s
NestedMultiple [19]	$max(n - x_0) + max(m - y_0)$	0.08s
SimpleMultipleDep [19]	$n \cdot (m + 1)$	0.15s
NestedMultipleDep [19]	$n \cdot (m + 1)$	0.09s
IsortList [21]	$n^2 \cdot m$	0.19s
LCS [21]	$n \cdot x$	0.15s
Example 1 [39]	n	0.15s
Sum [22]	$2n + 6$	0.15s
Flatten [22]	$8l + 8$	0.13s

(a) Representative examples of resource bound verification [19, 21, 22, 39], with non-linear and disjunctive bounds on running time (the upper part of the tabe) and heap space usage (the lower part of the table), as well as imperative and functional programs. #Horn execution times are slightly faster than the literature. All bounds are precise.

Program	**Bound**	**Time**
mcm	$\frac{(n+1)\cdot(n+2)}{2}$	0.6s
band matrix	$3n + 1$	0.8s

(b) Examples tracking relational dependencies between variables.

Leakage bound, bits	**Initialization**	**Time**
$log(1)$	$j = i$	1s
$log(\frac{n}{2})$	$j = i + \frac{n}{2}$	0.7s
$log(\frac{n}{3})$	$j = \frac{2 \cdot i + n}{3}$	0.7s

(c) Synthesis of countermeasures.

Table 1: Application of #Horn on three classes of examples.

Benchmarks from the literature We use #Horn to analyze a set of representative examples from the recent literature on resource bound computation (in particular: time and heap space), with results given in Table 1a. We find that all bounds derived by #Horn match those from the literature while being slightly faster on average.

On a technical level, we bound the time consumption of loops by synthesising a polytope that bounds the set of distinct tuples of loop indices. For example, for a loop with indices i and j bounded by parameters n and m, we synthesize a polytope the form: $a \leq i \leq n + b \wedge c \leq j \leq m + d$, where a, b, c, d are inferred by our method. For bounding heap consumption, we use the cost model of [22]. We encode *max* using disjunctions.

Examples requiring relational dependencies We use #Horn to analyze programs `mcm` for matrix chain multiplication of Section 2 and *band matrix* provided in Appendix D, with results in Table 1b. These examples require the tracking of relational dependencies between variables. The example `mcm` is particularly challenging as it requires reasoning about recursive function calls. We are not aware of any other method that can handle programs with both features. We use a template specifying that the polytope we would like to infer consists of three and four symbolic vertices, respectively. Note that choosing a template that is not expressive enough might only allow to prove a coarser bound, however, one can adress the problem of finding an appropriate template by running a loop over templates with an increasing number of symbolic vertices.

Synthesis of countermeasures By relying on recursive Horn clauses as input language, #Horn is readily applicable to a number of verification questions that go beyond reachability. We illustrate this using the example of procedure $\mathtt{find}(a, e)$, which returns the position of an element e in an array a. Note that the execution time of `find` (modeled by the variable t) reveals the position of e. We apply #Horn for synthesizing a padding countermeasure against this timing side channel. Namely, we seek to instantiate the initialization of the variable j such that it provides enough padding for a given bound on leakage. This is achieved by bounding the cardinality of the set of possible final values of t. We add an additional clause that constraints the cardinality of values for t upon termination, as the logarithm of this number corresponds to the amount of leaked information in bits, see e.g. [34]. Table 1c provides the timings and synthesized initialization of j for different bounds on leakage.

```
int find(a, e) {
 int r=-1; t=0;
 for(i=0; r<0 && i<n; i++)
   if (a[i]==e) {r=i; t++;}
 /* Padding */
 for(j=?; j<n, j++) t++;
}
/* assert: bound cardinality of
   set of final values of t. */
```

8 Related work

Counting integer points in polytopes The theory of counting integer points in polytopes has found wide-spread applications in program analysis. All applications we are aware of (including [2, 15, 26, 37]) compute cardinalities for given polytopes, whereas our interpolation method computes polytopes for given cardinality constraints.

Verdoolaege et al. [37] also derive symbolic expressions for the number of integer points in parametric polytopes. In their approach, the parameter governs only the offset of the bounding hyperplanes (and hence the position of the vertices of the polytope) but not their tilt (and hence not the generators of the vertex cones). The advantage of fixing the vertex cones is that Barvinok's decomposition can be applied to handle arbitrary polytope shapes. In contrast, our interpolation procedure $\#\textsc{Itp}_{\text{LIA}}$ (see Section 4) leaves the vertices *and* the generators of the vertex cones symbolic, up to constraints that ensure their unimodularity. The benefit of this approach is the additional degree of freedom for the synthesis procedure. #Horn leverages both approaches: the one from [37] for checking cardinality constraints, and $\#\textsc{Itp}_{\text{LIA}}$ for refining the abstraction.

Recently, [15] presented a logic and decision procedure for satisfiability in the presence of cardinality constraints for the case of linear arithmetic. In contrast, we focus on synthesizing formulas that satisfy cardinality constraints, rather than checking their satisfiability.

Resource bounds In [24] a static analysis estimates the worst case execution time of non-parametric loops using the box abstract domain. To compute precise bounds, the paper proposes a widening operator based on intersecting the current abstraction with polytopes derived from conditional statements. In contrast, our approach generates abstraction consisting of parametric unimodular polytopes (which include boxes as a special case). In [19], the authors compute parametric resource and time bounds by instrumenting the program with (multiple-) counters, using static analysis to compute a bound for the counters, and combining the results to yield a bound for the entire program. In contrast, we fit polytopes over each iteration domain of the program, thus avoiding the need to infer counter placement and enabling higher precision by tracking dependencies between variables. In [36] the authors propose a pattern-matching based method to extract polytopes representing the iteration domain of for-loops from C source. In contrast our method operates on unstructured programs represented as Horn clauses. In [22] and [21], a type system for the amortized analysis for higher-order, polymorphic programs is developed. Their focus lies on recursive data-types while we mostly deal with recursion/loops over the integers. In [1] and [28] the authors establish closed-form bounds on resource usage by solving recurrence relations over scalars.

Quantitative verification Existing verification methods for other theories rely on cardinality extensions of SAT [16], or Boolean algebra of (uninterpreted) sets [23], multisets [29], and fractional collections [30]. These approaches focus on either computing the model size or checking satisfiability of a formula containing cardinality constraints. Cardinalities of uninterpreted sets are also used in [18] for establishing termination and memory usage bounds based on fixed abstractions. Finally, a CEGAR approach for weighted transition systems has been presented in [10], together with abstractions for properties such as limit-average or discounted sum.

9 Conclusion

We applied the theory of counting integer points in polytopes to devise an algorithm for a cardinality-constrained extension of Craig interpolation. This algorithm proceeds by posing constraints on a symbolic polytope that represent both its shape and cardinality and then solves the constraints via state-of-the-art SMT solvers. We embedded our interpolation procedure into a solver for recursive Horn clauses with cardinality constraints and demonstrate its potential via an experimental evaluation.

Bibliography

[1] E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini. Cost analysis of java bytecode. In *Programming Languages and Systems*, pages 157–172. Springer, 2007.

[2] M. Backes, B. Köpf, and A. Rybalchenko. Automatic discovery and quantification of information leaks. In *IEEE S&P*, 2009.

[3] A. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. In *FOCS*, 1993.

[4] A. Barvinok. *A Course in Convexity*. American Mathematical Society, 2002.

[5] T. A. Beyene, S. Chaudhuri, C. Popeea, and A. Rybalchenko. A constraint-based approach to solving games on infinite graphs. In *POPL*, 2014.

[6] T. A. Beyene, C. Popeea, and A. Rybalchenko. Solving existentially quantified horn clauses. In *CAV*, 2013.

[7] N. Bjørner, K. L. McMillan, and A. Rybalchenko. Program verification as satisfiability modulo theories. In *SMT@IJCAR*, 2012.

[8] N. Bjrner, K. McMillan, and A. Rybalchenko. On solving universally quantified horn clauses. In *SAS*, 2013.

[9] M. Brion. Points entiers dans les polyedres convexes. *Ann. Sci. Ecole Norm. Sup.*, 21(4), 1988.

[10] P. Cerny, T. A. Henzinger, and A. Radhakrishna. Quantitative abstraction refinement. In *POPL*. ACM, 2013.

[11] M. Colón, S. Sankaranarayanan, and H. Sipma. Linear invariant generation using non-linear constraint solving. In *CAV*, 2003.

[12] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, second edition, 2001.

[13] J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida. Effective lattice point counting in rational convex polytopes. *J. of Symb. Comp.*, 38(4), 2004.

[14] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, 2008.

[15] M. Fredrikson and S. Jha. Satisfiability modulo counting: A new approach for analyzing privacy properties. In *LICS*. IEEE, 2014.

[16] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting. In *Handbook of Satisfiability*. 2009.

[17] S. Grebenshchikov, N. P. Lopes, C. Popeea, and A. Rybalchenko. Synthesizing software verifiers from proof rules. In *PLDI*, 2012.

[18] S. Gulwani, T. Lev-Ami, and M. Sagiv. A combination framework for tracking partition sizes. In *POPL*. ACM, 2009.

[19] S. Gulwani, K. K. Mehra, and T. Chilimbi. Speed: Precise and efficient static estimation of program computational complexity. In *POPL*, 2009.

[20] A. Gupta, C. Popeea, and A. Rybalchenko. Predicate abstraction and refinement for verifying multi-threaded programs. In *POPL*, 2011.

[21] J. Hoffmann, K. Aehlig, and M. Hofmann. Multivariate amortized resource analysis. In *POPL*, 2011.

[22] S. Jost, K. Hammond, H.-W. Loidl, and M. Hofmann. Static determination of quantitative resource usage for higher-order programs. In *POPL*, 2010.

[23] V. Kuncak, H. H. Nguyen, and M. C. Rinard. An algorithm for deciding BAPA: Boolean Algebra with Presburger Arithmetic. In *CADE*, 2005.

[24] P. Lokuciejewski, D. Cordes, H. Falk, and P. Marwedel. A fast and precise static loop analysis based on abstract interpretation, program slicing and polytope models. In *CGO*, 2009.

[25] L. Luu, S. Shinde, P. Saxena, and B. Demsky. A model counter for constraints over unbounded strings. In *PLDI*. ACM, 2014.

[26] P. Mardziel, S. Magill, M. Hicks, and M. Srivatsa. Dynamic enforcement of knowledge-based security policies. In *CSF*. IEEE, 2011.

[27] K. L. McMillan. An interpolating theorem prover. *TCS*, 2005.

[28] J. A. Navas, M. Méndez-Lojo, and M. V. Hermenegildo. User-definable resource usage bounds analysis for java bytecode. *Electr. Notes Theor. Comput. Sci.*, 253(5):65–82, 2009.

[29] R. Piskac and V. Kuncak. Decision procedures for multisets with cardinality constraints. In *VMCAI*, 2008.

[30] R. Piskac and V. Kuncak. Fractional collections with cardinality bounds, and mixed linear arithmetic with stars. In *CSL*, 2008.

[31] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In *VMCAI*, 2004.

[32] P. Rümmer, H. Hojjat, and V. Kuncak. Disjunctive interpolants for Horn-clause verification. In *CAV*, 2013.

[33] A. Schrijver. *Theory of linear and integer programming*. Wiley, 1999.

[34] G. Smith. On the Foundations of Quantitative Information Flow. In *FoSSaCS*, 2009.

[35] S. Verdoolaege. Barvinok. http://freecode.com/projects/barvinok.

[36] S. Verdoolaege and T. Grosser. Polyhedral extraction tool. In *IPACT*, 2012.

[37] S. Verdoolaege, R. Seghir, K. Beyls, V. Loechner, and M. Bruynooghe. Counting integer points in parametric polytopes using Barvinok's rational functions. *Algorithmica*, 2007.

[38] Wolfram. Wolfram alpha, series expansion. http://www.wolframalpha.com/examples/SeriesExpansions.html.

[39] F. Zuleger, S. Gulwani, M. Sinn, and H. Veith. Bound analysis of imperative programs with the size-change abstraction. In *SAS*, 2011.

A More on unimodularity

In this section, we give two alternative definitions of unimodularity.

Definition 1. *A cone is called unimodular if and only if its generators form a basis of* $\mathbb{Z}^d$.

Example 6. The cone given by generators $(0\ 1)$, $(1\ 0)$ is unimodular. In contrast, the cone given by generators $(1\ 2)$ and $(1\ 0)$ is not unimodular since e.g. $(1\ 1)$ cannot be represented as a positive linear combination of the generators.

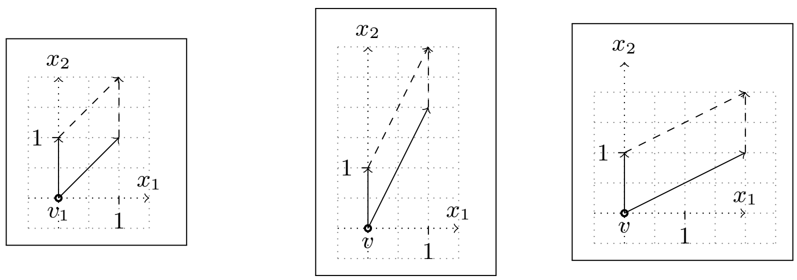


Fig. 2: Parallelepipeds for two unimodular cones and one non-unimodular cone. The last parallelepiped contains integer point $(1\ 1)$.

Equivalently, a cone is unimodular if and only if the parallelogram spanned by its generators contains only the origin. This parallelogram is called *parallelepiped*.

Definition 2. *The parallelepiped of a cone* K *with generators* $g_1, \ldots, g_k$ *is the set of points defined by*

$$\Pi_K = \{\textstyle\sum_{i=1}^{k} \alpha_i g_i \mid 0 \le \alpha_i < 1\} \ .$$

Then cone K is unimodular if and only if Π_K contains exactly one integer point, namely, the origin. We provide examples in Figure 2.

B Example: Parametric counting

Example 7. Consider polytope $Q = (x_1 \ge 0 \wedge x_2 \ge 0 \wedge x_1 + x_2 \le n)$, where the last equation is bounded by a parameter n rather than a constant. In this polytope, the coordinates of vertices v_2 and v_3 are linear expressions in the parameter n, that is, for $n \ge 0$ we have $v_2 = (0\ n)$ and $v_3 = (n\ 0)$. Equation (2) yields the following generating function.

$$\frac{x_1^0 x_2^0}{(1-x_1^0 x_2^1)(1-x_1^1 x_2^0)} + \frac{x_1^0 x_2^n}{(1-x_1^0 x_2^{-1})(1-x_1^1 x_2^{-1})} + \frac{x_1^n x_2^0}{(1-x_1^{-1} x_2^0)(1-x_1^{-1} x_2^1)}$$

Applying the substitution and computing the series expansion yields the constant coefficient $(n^2 + 3n + 2)/2$ which is an expression of number of integer points in Q in terms of the parameter n. ■

C Parametric Interpolation

In this section, we provide additional details on the parametric interpolation problem.

Example application of the interpolation algorithm

Example 8. Consider again the interpolation problem from Section 2. We assume the following template where we fix some of the coefficients for simplicity of presentation (our algorithm deals with the general case): $v_1 \mapsto \{H_1, H_4\}$, $v_2 \mapsto \{H_1, H_2\}$, $v_3 \mapsto \{H_2, H_3\}$ and $v_3 \mapsto \{H_3, H_4\}$ with $H_1 = -i \leq 0$, $H_2 = a \cdot j \leq n + b$, $H_3 = i \leq 1$ and $H_4 = i - j \leq 0$. We show exemplary vertex constraints for the parametric vertex $v_2 = (v_2^i \; v_2^j)$.

$$\forall n \exists v_2^i, v_2^j : a \cdot v_2^j = n + b \wedge v_2^i = 1 \wedge v_2^i > 0 \wedge v_2^i < v_2^j$$

Note that these vertex constraints are valid only for n such that $2 \leq (n + b)/a$, which is when v_2 is active in the polytope. To ensure this we add a constraint

$$\forall n : cmb(n) \rightarrow 2 \leq (n + b)/a.$$

We add corresponding constraints for the other vertices of the template and further require that $cmb(n)$ be implied by the lower bound φ^-.

Evaluating the generating function (as described in Section 3) then yields the following expression on the cardinality of φ in terms of a and b

$$\begin{aligned} \textsc{SymCard}(\varphi) = \quad & (1/2 - 1/(2a^2)) \cdot n^2 + (-b/a^2 + b/a + 1/a + 1) \cdot n + \\ & (1 + 2b/a) \end{aligned} \tag{4}$$

The cardinality constraint on φ is given by

$$\exists a, b \forall n : cmb(n) \rightarrow \textsc{SymCard}(\varphi) \leq \tfrac{(n+1)\cdot(n+2)}{2} \tag{5}$$

Solving the constraints yields $a = 1$, $b = 0$ and $cmb(n) = n \geq 2$. ■

Constrained based quantifier elimination method Consider Equation 5 which provides an example constraint that we would like to solve. Our technique builds on the following observation: 5 is equivalent to

$$\begin{aligned} \exists a, b : \forall n : \quad & cmb(n) \rightarrow \\ & 0 \leq ((1 - 2 \cdot c_2) \cdot n^2 + (3 - 2 \cdot c_1) \cdot n + (2 - 2 \cdot c_0)) \end{aligned} \tag{6}$$

where c_2, c_1 and c_0 denote the coefficients of n in Equation 4. Let $p(n)$ denote the polynomial in Equation 6.

For simplicity of presentation, assume that $p(n)$ is of full degree and therefore has exactly two roots r_1 and r_2. Then these roots induce a partitioning of the domain of $p(n)$ such that $p(n)$ is either positive or negative throughout each partition. To ensure that Equation 6 holds, we then have to ensure that whenever $cmb(n)$ holds, $p(n)$ is positive.

Exploiting the following equality which is a consequence of the factor theorem which states that each polynomial $p(n)$ with root r contains a factor $(n - r)$

$$p(n) = (n - r_1) \cdot (n - r_2) \cdot k = k \cdot n^2 - k \cdot (r_1 + r_2) \cdot n + k \cdot r_1 \cdot r_2$$

we can now obtain a symbolic representation of the roots by equating the coefficients of the two polynomials. This yields:

$$1 - 2 \cdot c_2 = k \wedge 3 - 2 \cdot c_1 = k \cdot (-r1 - r2) \wedge 2 - 2 \cdot c_0 = k \cdot r_1 \cdot r_2.$$

Note that this step is a source of incompleteness as it restricts the solution space to polynomials with roots that can be expressed in the respective theory, i.e. integers or reals. Then we ensure that $p(n)$ is positive whenever $cmb(n)$ holds through the following constraints

$$\begin{aligned} r1 \leq r2 \wedge ((cmb(n) \rightarrow n \leq r_1) \quad & \wedge 1 - 2 \cdot c_2 > 0) \ \vee \\ ((cmb(n) \rightarrow r_1 \leq n \leq r_2) & \wedge 1 - 2 \cdot c_2 < 0) \ \vee \\ ((cmb(n) \rightarrow n \geq r_2) \quad & \wedge 1 - 2 \cdot c_2 > 0). \end{aligned}$$

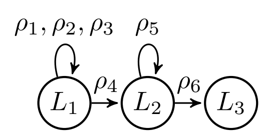
Here, we ensure positivity on the respective partition by referring to the concavity of $p(n)$ through its second derivative $p''(n) = 1 - 2 \cdot c_2$.

Note that the above constraints are quantifier free.

D Example: Verification conditions as Horn clauses

We consider a program that accesses a matrix stored in a dynamically allocated map m. The program manipulates the matrix through functions f and g. In the first loop, f is applied on a band around the diagonal, in the second loop, g is applied on the diagonal elements.

```
    int c1=-1; int c2=-1;
L1: for(i=0; i<n; i++)
      for (j=0; j=<i; j++)
        if (i-j<3) {
          m(i, j) = f(i, j);
          c1=i; c2=j;
        }
L2: for(i=0; i<n; i++) {
      v = m(i, i);
      m(i, i) = g(v, i);
      c1=i; c2=i;
    }
L3:
```



Our goal is to prove a bound on the memory consumption. To make the reasoning more explicit, we instrument the program with auxiliary variables c_1

and c_2 that store the pairs of indices used to write into the map. Thus, by reasoning about the cardinality of the set of values (c_1, c_2) we track the memory consumption of the program.

Let the program variables be given by the vector $v = (i, j, c_1, c_2, n, pc)$ (we do not track m, f, g, v for space reasons) and the initial states of the program be described by the assertion $init(v) = (i = 0 \wedge j = 0 \wedge c_1 = -1 \wedge c_2 = -1 \wedge pc = L_1)$. In the control flow graph above, we collapse the control locations for the nested loop into a single program point L_1.

Some relevant transition relations are described below (we omit equalities over variables that stay unchanged, e.g., $pc' = pc$).

$$
\begin{aligned}
\rho_1(v, v') &= (i < n \wedge j \leq i \wedge i - j < 3 \wedge j' = j + 1 \wedge c_1' = i \wedge c_2' = j) \\
\rho_2(v, v') &= (i < n \wedge j \leq i \wedge i - j \geq 3 \wedge j' = j + 1) \\
\rho_3(v, v') &= (i < n \wedge j > i \wedge j' = 0 \wedge i' = i + 1)
\end{aligned}
$$

We represent the bound verification condition as the following set of recursive Horn clauses over query symbols $\mathcal{Q} = \{reach, index\}$, where we let $c = (c_1, c_2)$ and i ranges between 1 and 6.

$$
\begin{array}{rll}
cl_{init}: & init(v) & \rightarrow reach(v) \\
cl_i \ : & reach(v) \wedge \rho_i(v, v') & \rightarrow reach(v') \\
cl_{proj}: & reach(v) \wedge c_1 \geq 0 \wedge c_2 \geq 0 & \rightarrow index(c, n) \\
cl_{card}: & n \geq 0 & \rightarrow |\{c \mid index(c, n))\}| \leq 3n + 1
\end{array}
$$

Query *reach* describes the set of reachable states and *index* describes the set of indices that were used for writing to the map. The clauses cl_{init}, and $cl_1, cl_2, \ldots$ require the invariant *reach* to be inductive, i.e., that is is implied by initial states and preserved under the transition relation. The clause cl_{proj} projects reachable states on variables c_1 and c_2, and ensures that all reachable values of c_1 and c_2 (except for the negative initial values) are included in *index*. The clause cl_{card} encodes a cardinality constraint stating that the cardinality of the set of index values is bounded by $3n + 1$. Finally, we note that the clauses are recursive, as e.g. cl_1 depends on itself.