A User's Guide for `LattE integrale` v1.7.2*

Velleda Baldoni, Nicole Berline, Jesús A. De Loera
Brandon Dutra, Matthias Köppe, Stanislav Moreinis
Gregory Pinto, Michèle Vergne, Jianqiu Wu

October 2014

*Research supported by NSF grants DMS-0309694, DMS-0073815, DMS-0914107, and DMS-0914873. Most of the students were supported by those grants and by summer fellowships provided through the UC Davis VIGRE grants DMS-0135345 and DMS-0636297. Image credit: Wikipedia.

Contents

1 Introduction

1.1 What is `LattE`?

The name “`LattE`” is an abbreviation for “**Latt**ice point **E**numeration.” `LattE` was developed in 2001 to count lattice points contained in convex polyhedra defined by linear equations and inequalities with integer coefficients. The polyhedra can be of any (reasonably small) dimension.

In 2007, `LattE macchiato` was released and contained many algorithmic improvements, in particular primal variants of the algorithms. The newest edition, `LattE integrale`, developed in 2010, can compute integrals of polynomials and volumes of rational polytopes. All these algorithms run in polynomial time for fixed dimension, or better.

`LattE integrale` was extended in 2012/2013 with a hybrid C++ and `Maple` implementation for computing the top coefficients of weighted Ehrhart quasipolynomials, using the general algorithm of [4], and the a C++ implementation of the algorithm specialized to knapsacks of [2, 1].

To learn more about the exact details of our implementation for lattice point enumeration, the interested reader can consult [14, 11, 10] and the references listed therein. For learning the algorithmic details of integration, see [3, 9]. Here we give a rather short description of the mathematical objects used by `LattE`. Note that all our computations are done over the integers or the rationals $e$xactly. `LattE` does not accept floating-point numbers as input.

1.1.1 Counting lattice points: Barvinok's Rational Functions

Given a convex polyhedron $P = \{u \in \mathbb{R}^d : Au \leq b\}$, where A and b are integral, the fundamental object that we compute is a short representation of the infinite power series:

$$f(P;x) \quad = \sum_{\alpha \in P \cap \mathbb{Z}^d} x_1^{\alpha_1} x_2^{\alpha_2} \dots x_d^{\alpha_d}.$$

Here each lattice point is given by one monomial. Note that this can be a rather long sum, in fact for a polyhedral cone it can be infinite, but the good news is that it admits short representations.

Example: Let P be the quadrangle with vertices $V_1 = (0,0)$, $V_2 = (5,0)$, $V_3 = (4,2)$, and $V_4 = (0,2)$, see Figure 1.

$$f(P;x,y) = x^5+x^4y+x^4+x^4y^2+yx^3+x^3+x^3y^2+yx^2+x^2+x^2y^2+xy+x+xy^2+y+1+y^2$$

The fundamental theorem of Barvinok (circa 1993, see [6]) says that you can write $f(P;x)$ as a sum of short rational functions, in polynomial time when the dimension of the polyhedron is fixed. In our running example we easily see that the 16 monomial polynomial can be written as shorter rational function sum:

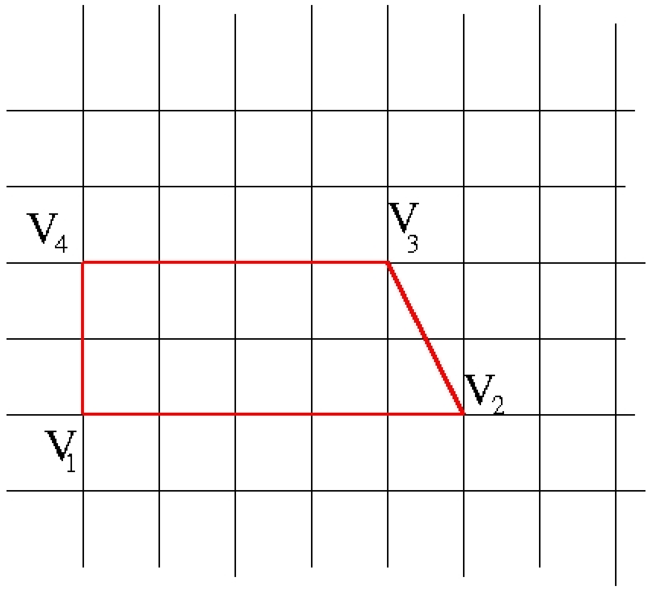


Figure 1: Quadrangle with vertices $V_1 = (0,0)$, $V_2 = (5,0)$, $V_3 = (4,2)$, and $V_4 = (0,2)$.

$$f(P;x,y) = f(K_{V_1};x,y) + f(K_{V_2};x,y) + f(K_{V_3};x,y) + f(K_{V_4};x,y)$$

where

$$f(K_{V_1};x,y) = \frac{1}{(1-x)(1-y)} \quad f(K_{V_2};x,y) = \frac{(x^5+x^4y)}{(1-x^{-1})(1-y^2x^{-1})}$$

$$f(K_{V_3};x,y) = \frac{(x^4y^2+x^4)}{(1-x^{-1})(1-xy^{-2})} \quad f(K_{V_4};x,y) = \frac{y^2}{(1-y^{-1})(1-x)}$$

$$f(P;1,1) = 16$$

Counting the lattice points in convex polyhedra is a powerful tool which allows many applications in areas such as Combinatorics, Statistics, Optimization, and Number Theory.

Fore details of how the computations are done, see [14, 11] .

1.1.2 Integration

`LattE integrale` has two different integration algorithms for integrating a rational polynomial $p \in \mathbb{R}[x_1, \ldots, x_d]$ over a d dimensional rational polytope. The first one, called the triangulation method, triangulates the polytope into simplices and integrates over each simplex. The other method, called the cone decomposition method, integrates over each tangent cone of the polytope. In order to do this, each tangent cone is triangulated into simple cones. This is the main trade off between the two integration algorithms: you can do one (possibly) large triangulation, or (possibly) many small tangent cone triangulations.

We decompose polynomials into finite sums of powers of linear forms because integrating powers of linear forms can be done in polynomial time [3]. Finding a decomposition of a polynomial as a sum of powers of linear forms is known as the polynomial Waring problem.

See [9] for a detailed explanation on why the next example gives the correct integral.

As an example, let us integrate the polynomial $x_1 + x_2$ over the unit square with vertices $(0,0),(1,0),(0,1)$ and $(1,1)$. The polynomial is already a power of a linear form so let $\ell = (1,1)$. To integrate $\int (x_1 + x_2)^M dx$ over the square, we need to compute

$$\frac{M!}{(M+d)!} |\det(u_1,\dots,u_d)| \frac{(\langle \ell, s\rangle)^{M+d}}{\prod_{i=1}^{d} \langle -\ell, u_i\rangle}$$

at each vertex s where the u_i are the rays from the tangent cone at s, and d is the dimension of the polytope.

Vertex $s_1 = (0,0)$: Because $\langle \ell, s_1\rangle^{1+2} = 0$ the valuation on this cone is zero.

Vertex $s_2 = (1,1)$:

$$\frac{M!}{(M+d)!} |\det(u_1,\dots,u_d)| \frac{(2)^{1+2}}{(-1)(-1)} = \frac{1!}{(1+2)!} \times 1 \times 8 = 4/3$$

Vertex $s_3 = (1,0)$:

$$\frac{M!}{(M+d)!} |\det(u_1,\dots,u_d)| \frac{(1)^{1+2}}{(1)(-1)} = \frac{1!}{(1+2)!} \times 1 \times -1 = -1/6$$

Vertex $s_4 = (0,1)$:

$$\frac{M!}{(M+d)!} |\det(u_1,\dots,u_d)| \frac{(1)^{1+2}}{(1)(-1)} = \frac{1!}{(1+2)!} \times 1 \times -1 = -1/6$$

The integral $\int_{x_1=0}^{x_1=1} \int_{x_2=0}^{x_2=1} x_1 + x_2 \, \mathrm{d}x_1 \, \mathrm{d}x_2 = 0 + 4/3 - 1/6 - 1/6 = 1$ as it should be.

1.1.3 Weighted Ehrhart Quasi-Polynomials

`LattE integrale` can also compute the weighted Ehrhart quasipolynomials where the weight function is a polynomial or a sum of powers of linear forms. See [4]. This functionality requires `Maple`.

1.1.4 Ehrhart Quasi-Polynomials of Knapsack polytopes

`LattE integrale` can compute the Ehrhart quasipolynomials as function of t of polytopes in the form

$$\{x \in \mathbb{R}^n_{\geq 0} \mid \alpha_1 x_1 + \cdots + \alpha_n x_n = t\}.$$

These polytopes are also related to Sylvester's denumerant. See [2, 1].

1.2 Which programs of `LattE` compute what?

`LattE` contains three key executables:

`count` counts lattice points, computes Ehrhart polynomials and Ehrhart series of polytopes. This executable has replaced `ehrhart`, but `ehrhart` is still included for backwards compatibility.

`integrate` integrates polynomials, powers of linear forms, and products of powers of linear forms over polytopes. Integrate can also computed weighted Ehrhart quasipolynomials.

`top-ehrhart-knapsack` Computes the Ehrhart quasipolynomials for knapsack-type polytopes.

`latte-maximize`, `latte-minimize` perform linear integer optimization.

The other executables in latte are drivers, converters, and other small utility functions.

1.3 Maple programs distributed with `LattE`

`LattE integrale` also comes with a number of Maple programs, which can be used independently of the main `LattE` code. We distribute them for several reasons:

- As a pedagogical tool, as it is often easier to understand the straightforward version of the algorithm written in Maple than it is to understand the optimized C++ implementation in `LattE`.
- As a technology preview, for some advanced algorithms that have not yet been implemented in C++.
- For cross-checking the correctness of our C++ implementations.

The Maple programs come from the LattE source directory `code/maple`. They are installed in `dest/share/latte-int/`.

The following `README` file gives more information.

```
This directory, available both on the LattE website and as part of the
LattE integrale distribution (in the directory code/maple), contains
LattE's Maple programs, which can be used independently.


Conebyconeapproximations_08_11_2010.mpl

  It contains Maple code for computing the highest coefficients of
  Ehrhart quasi-polynomials, using the algorithm of this paper:

  - Velleda Baldoni, Nicole Berline, Jesus A. De Loera, Matthias
    Koeppe, and Michele Vergne, Computation of the highest
    coefficients of weighted Ehrhart quasi-polynomials of rational
    polyhedra, Foundations of Computational Mathematics 12 (2012),
    435-469, doi:10.1007/s10208-011-9106-4

  These functions are also accessible using LattE's "integrate"
  function; see the LattE manual.

  It can also compute the canonical cone-by-cone patched
  quasi-polynomial defined in the paper:

  - Velleda Baldoni, Nicole Berline, Jesus A. De Loera, Matthias
    Koeppe, and Michele Vergne, Three Ehrhart Quasi-polynomials, 2014.


m-knapsack.mpl

  It contains Maple code for computing the highest coefficients of
  Ehrhart quasi-polynomials of knapsack polytopes, using the algorithm
  of the papers:

  - Velleda Baldoni, Nicole Berline, Jesus A. De Loera, Brandon
    E. Dutra, Matthias Koeppe, and Michele Vergne, Top degree
    coefficients of the denumerant, 25th International Conference on
    Formal Power Series and Algebraic Combinatorics (FPSAC 2013),
    DMTCS proc. AS, Discrete Mathematics and Theoretical Computer
    Science (DMTCS), 2013, pp. 1149-1160, available from
    http://www.dmtcs.org/dmtcs-ojs/index.php/proceedings/article/view/dmAS0197/4324

  - Velleda Baldoni, Nicole Berline, Jesus A. De Loera, Matthias
    Koeppe, and Michele Vergne, Coefficients of Sylvester's denumerant,
    eprint arXiv:1312.7147 [math.CO], 2013

  LattE integrale also contains a native C++ implementation of the
  same algorithm.  It is MUCH faster, so we recommend that for any
```

```
  production use.  See the LattE manual.


RealBarvinok-mars-exemples-2014-03-10.mpl

  It contains Maple code for computing intermediate generating
  functions S^L (by means of Brion-Vergne decomposition) and
  intermediate Ehrhart quasi-polynomials, using the algorithms in:

  - Velleda Baldoni, Nicole Berline, Matthias Koeppe, and Michele
    Vergne, Intermediate sums on polyhedra: Computation and real
    Ehrhart theory, Mathematika 59 (2013), no. 1, 1-22,
    doi:10.1112/S0025579312000101

  It also contains Maple code for computing the canonical Barvinok
  patched generating function and quasi-polynomial for a simplex,
  as described in:

  - Velleda Baldoni, Nicole Berline, Jesus A. De Loera, Matthias
    Koeppe, and Michele Vergne, Three Ehrhart Quasi-polynomials, 2014.


3-ehrhart-polynomials-paper-examples.mpl

  Computational examples from the paper:

  - Velleda Baldoni, Nicole Berline, Jesus A. De Loera, Matthias
    Koeppe, and Michele Vergne, Three Ehrhart Quasi-polynomials, 2014.


The other files are part of LattE's build system and can be ignored.
```

2 Input Files

A polytope can be defined from a list of vertices (a v-representation) or a list of hyperplane inequalities (h-representation) and so `LattE` can start from either representation in different formats. Here are four common file formats:

1. `LattE` style vertex file
2. `LattE` style hyperplane file
3. `CDD` style vertex file
4. `CDD` style hyperplane file

Users of `Polymake` will notice that `Polymake`'s facets and vertices are printed in a format that is easily converted to a `LattE` style h- or v-representation.

LattE comes with a large library of example input files compiled from various sources. In the source distribution, it can be found in the EXAMPLES subdirectory; after installation, it can be found in dest/share/latte-int/examples.

We now explore the file syntax of each.

2.1 LattE h-representation

2.1.1 Inequality Description

Let P be a polytope described by a system of inequalities $Ax \leq b$, where $A \in \mathbb{Z}^{m \times d}$, $A = (a_{ij})$, and $b \in \mathbb{Z}^m$. Note that any hyperplane representation with rational coefficients can be brought into this form; for example $x + 1/2y \leq 5/9$ should be written as $18x + 9y \leq 10$. With $P = \{x : Ax \leq b\}$, the input file is;

```
m d+1
b  -A
```

Example: Let $P = \{(x, y) : x \leq 1, y \leq 1, x + y \leq 1, x \geq 0, y \geq 0\}$. Thus

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \quad , \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

and the LattE input file would be

```
5 3
1 -1  0
1  0 -1
1 -1 -1
0  1  0
0  0  1
```

2.1.2 Equality Constraints

By default, a constraint is an inequality of type $a^T x \leq b$. But to input an equality constraint $a^T x = b$ we need to add a keyword.

Example: Let P be as in the previous example, but require $x + y = 1$ instead of $x + y \leq 1$, thus, $P = \{(x, y) : x \leq 1, y \leq 1, x + y = 1, x \geq 0, y \geq 0\}$. Then the LattE input file that describes P would be as such:

```
5  3
1 -1  0
1  0 -1
1 -1 -1
0  1  0
0  0  1
linearity 1 3
```

The last line states that among the 5 inequalities one is to be considered an equality, the third one.

In general, the linearity syntax is :

```
linearity <number of equations> <row indices...>
```

The row indices start at 1.

2.1.3 Nonnegativity Constraints

For bigger examples it quickly becomes cumbersome to state all nonnegativity constraints for the variables one by one. Instead, you may use another shorthand.

Example: Let P be as in the previous example, then the `LattE` input file that describes P could also be described as such:

```
3  3
1 -1  0
1  0 -1
1 -1 -1
linearity 1 3
nonnegative 2 1 2
```

The last line states that there are two nonnegativity constraints and that the first and second variables are required to be nonnegative. **NOTE** that the first line reads "3 3" and not "5 3" as above!

In general, the nonnegative syntax is :

```
nonnegative <number of variables in list> <variable indices...>
```

The variable indices start at 1.

2.1.4 Cost Vector

The functions maximize and minimize solve the integer linear programs

$$\max\{c^{\intercal}x : x \in P \cap \mathbb{Z}^d\}$$

and

$$\min\{c^{\intercal}x : x \in P \cap \mathbb{Z}^d\}.$$

Besides a description of the polyhedron P, these functions need a linear objective function given by a certain cost vector $c \in \mathbb{Z}^d$, where the input style is very similar to a `LattE` h-representation file.

Example: If the polyhedron is given in the file "fileName"

```
4  4
1 -1  0  0
1  0 -1  0
1  0  0 -1
1 -1 -1 -1
linearity 1 4
nonnegative 3 1 2 3
```

the cost vector must be given in the file "fileName.cost", as for example in the following three-dimensional problem:

```
1 3
2 4 7
```

The first two entries state the size of a $1 \times n$ matrix (encoding the cost vector), followed by the $1 \times n$ matrix itself. Assuming that we call maximize, this whole data encodes the integer program

$$\max\{2x_1 + 4x_2 + 7x_3 : x_1 + x_2 + x_3 = 1, x_1, x_2, x_3 \in \{0, 1\}\}.$$

2.2 `LattE` v-representation

`LattE` can start from a homogenized v-representation of the polytope. To homogenize a vertex, simply add an leading 1 to the vertex. This has the effect of lifting the polytope to a cone in one dimension higher such that the original polytope can be extracted by intersecting the cone with the $x_1 = 1$ plane. For example, take a triangle in the plane, then Figure 2 shows the resulting cone.

Let $v_1, \ldots, v_k$ be the vertices of a polytope $P \subseteq \mathbb{R}^n$, then the `LattE` v-representation file format is:

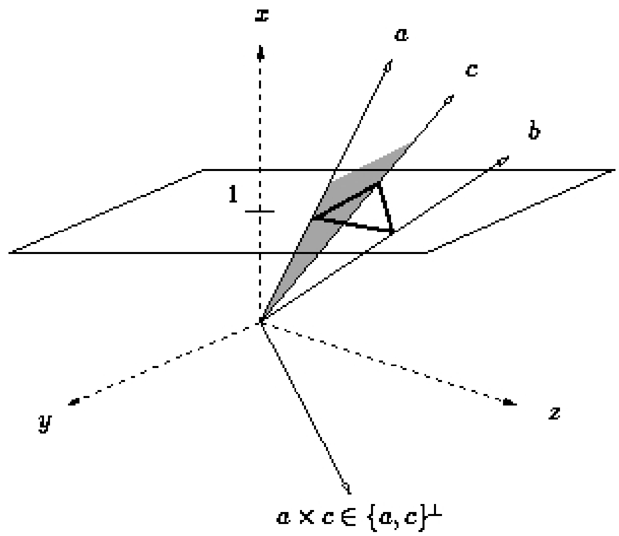


Figure 2: Homogenized triangule.

$$
\begin{array}{l}
k\ (n+1) \\
1\ v_1 \\
\vdots \\
1\ v_k
\end{array}
$$

Example: Note, like `LattE` h-representations files, a rational-vertex polytope with can be written with integer data by scaling each homogenized vertex. Below are the vertices of a rectangle $(0,0), (2/3,0), (0,1/4), (2/3,1/4)$:

```
4 3
1 0 0
3 2 0
4 0 1
12 8 3
```

2.3 CDD Input Files

In addition to the formats described above, `LattE` can also accept input files in standard `CDD` format. Below is an example of `CDD` input that is readable into `LattE`.

```
H-representation
begin
4  4  integer
2 -2  4 -1
3 -2 -2  3
6  2 -4 -3
1  2  2  1
end
```

For a complete description of `CDD` file syntax, see the `CDD` manual [12]. Pass the command-line option `--cdd` to `LattE` if you use a `CDD` input format.

2.4 Non–full dimensional polytopes

When the input polytope is not full dimensional, `LattE` projects that polytope such that it becomes full dimensional. This transformation preserves the lattice point count of the input polytope.

Note, however, that this is not supported when the input is given in LattE v-representation.

Also note that the integration routines do not currently support non–full dimensional polytopes.

2.5 `LattE` vs. `CDD` file formats

There are a few key differences between `LattE` and `CDD` file formats.

1. `CDD` uses the file extension `*.ine` for h-representation files, and `*.ext` for v-representation files. However, `LattE` makes no assumption on the file extensions of files. This manual has recommended using `*.vrep.latte` and `*.hrep.latte` for `LattE` style files, but you are free to name your files anything; even our own example files do not follow this convention.

2. `CDD` also requires "H-representation" or "V-representation" keywords in the file. Pass the command-line option `--cdd` to `LattE` if you use either of the two `CDD` formats.

3. On the other hand, forgetting about the "linearity" and "nonnegative" keywords, there is no syntactic difference between a `LattE` v- and h-representation file. Therefore, you need to provide the command-line option `--vrep` whenever you input a `LattE` v-representation file.

2.6 Polynomials and linear forms

`LattE integrale` can also integrate polynomials and in particular sums of powers of linear forms over polytopes. Powers of linear forms are the fundamental structure used to integrate. Next, we describe the syntax of polynomials and linear forms

- A polynomial is represented as a list of its monomials in the form

  [*monomial*$_1$,*monomial*$_2$,...],

  where *monomial*$_i$ is represented by

  [*coefficient*,[*exponent-vector*]].

  For example, $3x_0^2x_1^4x_2^6 + 7x_1^3x_2^5$ is input as `[[3,[2,4,6]], [7,[0,3,5]]]`.

- To deal directly with sums of powers of linear forms, a fundamental data structure in `LattE integrale`, the input format is

  [*linear-term*$_1$, *linear-term*$_2$, ...],

  where *linear-term*$_i$ is represented by

  [*coefficient*,[*power*,[*coefficient-vector*]]].

  For example, $3(2x_0+4x_1+6x_2)^{10}+7(3x_1+5x_2)^{12}$ is input as `[[3,[10,[2,4,6]]], [7,[12,[0,3,5]]]]`.

The reason this is useful is because any polynomial can be written as a sum of powers of linear forms, see [3].

2.7 Products of linear forms

`LattE integrale` can integrate a product of linear forms over a simplex or a triangulation of a polytope.

The input format for a sum of products of linear forms is

[[*coefficient*, [[*power*$_1$, [*coefficient-vector*$_1$]],[*power*$_2$, [*coefficient-vector*$_2$]], ...]], ...].

For example, the integrand

$$(1x_1 + 2x_2)^3(4x_1 + 5x_2)^6 + 7(8x_1 + 9x_2)^{10}(11x_1 + 12x_2)^{13}(14x_1 + 15x_2)^{16}$$

is written as

```
[[1, [ [3, [1, 2]], [6, [4, 5]] ]], [7, [ [10, [8, 9]], [13, [11, 12]],
[16, [14, 15]] ]] ]
```

3 Running LattE

The executables of LattE integrale are installed in the "bin/" subdirectory of the installation tree. The standard distribution of LattE integrale sets the installation tree to be the "dest/" subdirectory of the source tree. Thus, to invoke the count executable, you would type

```
dest/bin/count
```

3.1 How to use count

count has a nice help menu, to view it, run

```
count --help
```

The following options control what count computes.

- Count the number of lattice points in polytope P, where P is given in a file named "fileName.hrep.latte" in different file formats.

  ```
  count fileName.hrep.latte
  count --vrep fileName.vrep.latte
  count --cdd fileName.ine
  ```

- Count the number of lattice points in nP, the dilation of P by the integer factor n.

  ```
  count --dilation=n fileName.hrep.latte
  ```

- Use the homogenized Barvinok algorithm [8] to count the number of lattice points in the polytope P. Use if number of vertices of P is big compared to the number of constraints.

  ```
  count --homog fileName.hrep.latte
  ```

- Compute the number of lattice points (default)

  ```
  count --count-lattice-points  fileName.hrep.latte
  ```

- Compute the multivariate generating function of the set of lattice points of the polyhedron

```
count --multivariate-generating-function
    fileName.hrep.latte
```

For unbounded polyhedra, one needs to combine this with the option `--compute-vertex-cones=4ti2` (now the default when 4ti2 is available), as other methods in `LattE` currently refuse to handle unbounded polyhedra. For example,

```
count --compute-vertex-cones=4ti2
      --multivariate-generating-function fileName.hrep.latte
```

writes the multivariate generating function (in Maple notation) to "fileName.rat."

- Compute the Ehrhart polynomial of a lattice polytope

  ```
  count --ehrhart-polynomial  fileName.hrep.latte
  ```

  Note: For the computation of weighted Ehrhart quasipolynomials of rational polytopes, or their top coefficients, see section 3.2.1 below.

- Compute the unsimplified Ehrhart series as a univariate rational function

  ```
  count --ehrhart-series  fileName.hrep.latte
  ```

- Compute the simplified Ehrhart series as a univariate rational function (needs `Maple`).

  ```
  count --simplified-ehrhart-series  fileName.hrep.latte
  ```

- Compute the first N terms of the Ehrhart series

  ```
  count --ehrhart-taylor=N  fileName.hrep.latte
  ```

The following options relate to the Barvinok algorithm and were introduced by Matthias Köppe in `LattE macchiato`, see [14]. Not all modes of operation support all options.

- Triangulate and signed-decompose in the dual space (traditional method, default)

  ```
  count --dual  fileName.hrep.latte
  ```

- Triangulate in the dual space, signed-decompose in the primal space using irrationalization

  ```
  count --irrational-primal  fileName.hrep.latte
  ```

- Triangulate and signed-decompose in the primal space using irrationalization

  ```
  count --irrational-all-primal  fileName.hrep.latte
  ```

- Decompose cones down to an index (determinant) of N instead down to unimodular cones (which have an index of 1).

  ```
  count --maxdet=N  fileName.hrep.latte
  ```

- Do not signed-decompose simplicial cones

  ```
  count --no-decomposition   fileName.hrep.latte
  ```

- Use polynomial substitution for specialization (traditional method, default)

  ```
  count --polynomial  fileName.hrep.latte
  ```

- Use exponential substitution for specialization (recommended for maxdet larger than 1)

  ```
  count --exponential   fileName.hrep.latte
  ```

REMARK The functionality of the `LattE` v1.2 `ehrhart` command has been merged into `count`:

```
count --ehrhart-series FILENAME

      (replaces: ehrhart FILENAME)

count --simplified-ehrhart-series FILENAME

      (replaces: ehrhart simplify FILENAME)

count --ehrhart-taylor=N FILENAME

      (replaces: ehrhart N FILENAME)
```

The `ehrhart` program is still available, but it does not accept the new command-line options of `count`.

3.2 How to use `integrate`

Like `count`, `integrate` has a help menu. To view the menu, run

```
integrate --help
```

There are two different integration (and volume) algorithms. The triangulation method triangulates the entire polytope and integrates over each simplex. In the cone decomposition method we integrate over each cone, possibly triangulating it first. Unlike other integration software, `LattE` integrates polynomials and powers of linear forms in exact arithmetic.

`integrate` is also able to compute weighted Ehrhart quasi-polynomials, see the next subsection.

- Integrates using the cone-decomposition method.

  ```
  --cone-decompose
  ```

- Integrates using the triangulation method.

  ```
  --triangulate
  ```

- Sets what you want to compute: a volume or an integral.

  ```
  --valuation=integrate
  --valuation=volume
  ```

- Sets the file that contains the polynomial, powers of linear forms, or products of powers of linear forms. If this option is not set, and the valuation is integration, the integrand will be read from stdin.

  ```
  --monomials=FILE
  --linear-forms=FILE
  --product-linear-forms=FILE
  ```

  If the integrand is a polynomial or a power of a linear form, there are two integration and volume algorithms available: a polytope triangulation based method and a tangent cone based method. If the integrand is a product of powers of linear forms, there is only one algorithm available and it is a polytope triangulation based method.

Example: Let us view a few examples of the above options

- Integrates a polynomial in file "FILE" using the triangulation method.

```
integrate --valuation=integral --triangulate
  --monomials=FILE fileName.hrep.latte
```

- Find a volume using the cone decomposition method from a `LattE` v-representation file.

```
integrate --valuation=volume --cone-decompose
  --vrep fileName.vrep.latte
```

- If an integration method is not given, `LattE integrale` computes the integral with *both* methods. This can also be done by the `--all` option. The next two commands do the same thing: find a volume using both methods from a `LattE` v-representation file.

```
integrate --valuation=volume --vrep fileName.vrep.latte

integrate --valuation=volume --all
  --vrep fileName.vrep.latte
```

3.2.1 How to use `integrate` for computing (highest coefficients of) weighted Ehrhart quasipolynomials

This option currently requires `Maple`. For non-lattice polytopes, the coefficients of the Ehrhart quasipolynomials are computed as step functions. An amazing feature of this algorithm is that **we can compute a small number of highest coefficients of the Ehrhart quasipolynomial even for polytopes of rather high dimension**, for which the computation of the full quasipolynomial is completely out of reach. See [4] for the details, including a discussion of the complexity; this paper is based on the pioneering work in [5].

The triangulation of vertex cones is performed in the dual space; command line options to do otherwise are ignored.

By default, `LattE integrale` computes the coefficients incrementally, starting from the highest coefficient, which is the easiest to compute. You can interrupt the computation at any time if you don't need any further coefficients. If you know in advance how many coefficients from the top you need, use the option `--num-coefficients=K`. This will be slightly faster than incremental computation.

- Sets what you want to compute: weighted Ehrhart quasipolynomials.

```
--valuation=top-ehrhart
```

- Sets the weight function from a file.

```
--monomials=FILE
--linear-forms=FILE
```

- Sets the weight function to 1 (unweighted). This is the default.

```
--top-ehrhart-unweighted
```

- Only compute the top K coefficients. The quasipolynomial's coefficients are not computed incrementally. Use this option if you know in advance you can wait for the software to finish or have low memory requirements. When this option is missing, the entire quasipolynomial's coefficients are computed incrementally which takes more memory but you may manually stop the computation at any time.

```
--num-coefficients=K
```

- Save the quasipolynomial to a file. If "–num-coefficients=K" is used, the file write is done after the quasipolynomial is computed. If "–num-coefficients=K" is missing, the quasipolynomial's coefficients are saved as they are computed.

```
--top-ehrhart-save=FILE
```

- Compute the weighted Ehrhart polynomial that is valid for non-integer dilations, rather than integer dilations only. The formulas for the coefficients will usually be more complicated. Even for lattice polytopes, one obtains Ehrhart quasipolynomials rather than Ehrhart polynomials.

```
--real-dilations
```

- Sets interactive mode if you want to manually type a polynomial or a sum of linear forms. You cannot compute a weighted Ehrhart quasipolynomial where the weight is a product of linear forms.

```
--interactive-mode
```

Example: Let us view a few examples of the above options.

- Compute the unweighted Ehrhart quasipolynomial that is valid for real dilations, rather than integer dilations only.

```
integrate --valuation=top-ehrhart --real-dilations
          fileName.hrep.latte
```

- Compute weighted Ehrhart quasipolynomial where the weight is a polynomial.

```
integrate --valuation=top-ehrhart --monomials=FILE
          fileName.hrep.latte
```

- Find only the two largest degree terms of the linear form weighted Ehrhart quasipolynomial

```
integrate --valuation=top-ehrhart --linear-forms=FILE
          --num-coefficients=2 fileName.hrep.latte
```

- Manually enter a weight function and save the Ehrhart quasipolynomial to a file

```
integrate --valuation=top-ehrhart --interactive-mode
          --top-ehrhart-save=FILE fileName.hrep.latte
```

Example: A concrete example using a file from the library of examples that comes with the LattE distribution. We count the lattice points in the 3-dimensional cube $[-1,1]^3$, dilated by a *real* dilation factor N.

```
integrate --valuation=top-ehrhart --cdd --real-dilations
          --top-ehrhart-save=cube_3_real.txt
          dest/share/latte-int/examples/cubes/cube_3.ext
```

This creates a file `cube_3_real.txt` with the following content (the formulas are long; some parts have been elided here):

```
epoly:= \
+ 8*N^3\
+ (12-24*MOD(n,1))*N^2\
+ (6-12*MOD(n,1)+359/125245152*(-5521248*MOD(n,1)+2760624)*MOD(n,1)-[...])*N\
+ 1-1700/431*MOD(n,1)+1652/431*MOD(n,1)^2-[...];
```

As this example shows, the quasi-polynomial is expressed as a polynomial in N (the dilation factor), whose coefficients are step-polynomials. The polynomial degree in N is 3 (the dimension of the polytope). The step-polynomials are expressed using the fractional-part operation $\mathrm{MOD}(x, 1)$, using the variable n (again the dilation factor). (Having two different variables for the dilation factor makes it easier to manipulate this expression in computer algebra systems such as Maple.)

This formula is valid for arbitrary real (not just rational) dilations. See below for how to conviently evaluate it in `Maple`.

Example: For comparison, let's compute the same example without the `--real-dilations` parameter.

```
integrate --valuation=top-ehrhart --cdd
          --top-ehrhart-save=cube_3_integer.txt
          dest/share/latte-int/examples/cubes/cube_3.ext
```

This creates a file `cube_3_integer.txt` with the following content:

```
epoly:= \
+ 8*N^3\
+ 12*N^2\
+ 6*N\
+ 1;
```

Since the input was a lattice polytope, this is an Ehrhart polynomial (all coefficients are constants).

Example: Another example. This time we ask only for the highest 3 coefficients of the Ehrhart quasi-polynomial, valid for real dilations, of a 6-dimensional simplex from the library, which is given in `LattE` vrep format.

```
integrate --redundancy-check=none --triangulation=4ti2
          --valuation=top-ehrhart --num-coefficients=3
          --top-ehrhart-save=random-simplex-real-top3.txt
          --vrep --real-dilations
          dest/share/latte-int/examples/random-simplex/random-simplex-dim6-digits4-
```

This computation takes a while and creates a file `random-simplex-real-top3.txt`. It is large, and we don't show it here.

Example: The same example, but we do not know in advance how many coefficients we want. We do have a limited budget of time or patience. So we ask for an incremental computation, starting at the top coefficients. We can interrupt at any time by typing CONTROL-C.

```
integrate --redundancy-check=none --triangulation=4ti2
          --valuation=top-ehrhart
          --top-ehrhart-save=random-simplex-real-top3.txt
          --vrep --real-dilations
          dest/share/latte-int/examples/random-simplex/random-simplex-dim6-digits4-1
```

If interrupted after the term of N^5 is shown on the screen, the file `random-simplex-real-incremental.txt` looks like this:

```
epoly:= \
+ 15181275198303665635391/360*N^6\
+ (-1/120*MOD(-64173062910517600471035*n,1)-[...])*N^5
```

To facilitate reading this file into `Maple`, let's add a semicolon at the end.

```
echo ";" >> random-simplex-real-incremental.txt
```

3.2.2 Evaluating Ehrhart quasi-polynomials in `Maple`

LattE's `Maple` library has a helper function called `evaluateEhrhart` that is useful for evaluating quasi-polynomials in this form. You can find it in file `dest/share/latte-int/Conebyconeapproximations_08_11_2010.mpl`. Let's look at a sample `Maple` session. We will refer to the files created in the previous section. First load the code:

```
> read "dest/share/latte-int/Conebyconeapproximations_08_11_2010.mpl":
```

Now read in one of the example Ehrhart quasi-polynomials.

```
> read "cube_3_real.txt";
```

`Maple` displays the quasi-polynomial. It is always assigned to variable `epoly`. The output is long, but still fits on the screen (if your screen is big enough). Let's evaluate it for a few integer and rational dilations:

```
> evaluateEhrhart(epoly, 0);
                                      1

> evaluateEhrhart(epoly, 1);
                                     27

> evaluateEhrhart(epoly, 3/2);
                                     27
```

```
> evaluateEhrhart(epoly, 2-1/1000000);
                                   27

> evaluateEhrhart(epoly, 2);
                                   125
```

These are all exact answers. Now let's try real dilations. `evaluateEhrhart` allows you to enter floating point numbers. Remember, `Maple` represents all entered decimal fractions exactly, using floating point with basis 10. `evaluateEhrhart` interprets a given floating point number as an interval that represents that all given digits are correct, but the last given digit may be rounded:

```
> floatToInterval(2.0);
                                 39  41
                                 --, --
                                 20  20
> floatToInterval(2.00);
                                399  401
                                ---, ---
                                200  200
```

If the dilation factor is given with enough precision, then by analyzing this interval, we can evaluate the quasi-polynomial *exactly* (the answer will be an integer (or, in the weighted case, a rational number):

```
> evaluateEhrhart(epoly, 2.3);
# Exact answer (assuming that all provided digits of the
# floating-point dilation factor were correct):
                                   125
```

However, if you try this for a dilation where there is a jump discontinuity in the quasi-polynomial, this method cannot work, and we return a simple floating point evaluation of the formula. Use the result with caution.

```
> evaluateEhrhart(epoly, 2.0);
# The precision of the given floating point dilation factor is
# not large enough to allow exact computation.  Resorting to
# floating point evaluation.  Increase Digits if you want more
# precision in this evaluation.
                                 125.000
```

We can also evaluate at symbolic expressions. However, `evaluateEhrhart` has to rely on the correctness of `Maple`'s `floor` function for symbolic arguments. Also, the system variable `Digits` needs to be set to a large enough value, or `Maple` will leave some `floor` expressions unevaluated.

```
> Digits := 4: evaluateEhrhart(epoly, 707*sqrt(2));
                       1/2                     1/2 2                    1/2 3
      1 + 6 floor(707 2   ) + 12 floor(707 2   )  + 8 floor(707 2   )

> Digits := 8: evaluateEhrhart(epoly, 707*sqrt(2));
                                 7988005999
```

Let's switch to another example.

```
> read "random-simplex-real-incremental.txt":
```

The Ehrhart quasi-polynomial, valid for real dilations, has a complicated structure with many jump discontinuities. Any evaluation at an integer or rational number will be exact.

```
> evaluateEhrhart(epoly, 1234/1000);

                   167513276311474881671845822495473290521
                   ---------------------------------------
                             1125000000000000000
```

If we try to give a floating point number, however, we need to give it with really high precision; but even then, we cannot evaluate it exactly because we only computed the top 3 coefficients of the quasi-polynomial. So the best we can do is a floating-point approximation.

```
> evaluateEhrhart(epoly, 1.23);
# The precision of the given floating point dilation factor is
# not large enough to allow exact computation.  Resorting to
# floating point evaluation.  Increase Digits if you want more
# precision in this evaluation.
                                            21
                             0.14602810 10

> evaluateEhrhart(epoly, 1.23456789098765);
# The precision of the given floating point dilation factor is
# not large enough to allow exact computation. [...]
                                            21
                             0.14931232 10

> evaluateEhrhart(epoly, 1.23456789098765345678976543456789);
# Given Ehrhart quasi-polynomial was not complete; evaluating
# using floating point.  Increase Digits if you want more precision.
                                            21
                             0.14931233 10
```

```
> Digits := 40: evaluateEhrhart(epoly, 1.23456789098765345678976543456789);
# Given Ehrhart quasi-polynomial was not complete; evaluating
# using floating point.  Increase Digits if you want more precision.
                                                                  21
               0.1493123109910186285550465481092293593932 10
```

Symbolic expressions work better in this case. Again, the system variable `Digits` needs to be set to a large enough value, or `Maple` will leave some `floor` expressions unevaluated.

```
> Digits:=4: evaluateEhrhart(epoly, exp(1));
 15181275198303665635391       6
------------------------ exp(1)
           72

                   5
     + 1/120 exp(1)  floor(-64173062910517600471035 exp(1))

                   5
     + 1/120 exp(1)  floor(-32059652032188669736011 exp(1))

     [...]

> Digits:=40: evaluateEhrhart(epoly, exp(1));
   15181275198303665635391       6   20633492252192421384473       5
  ------------------------ exp(1)  + ----------------------- exp(1)
             72                                30
```

3.3 Options common to both `count` and `integrate`

A common subproblem in counting lattice points and integration requires finding triangulations and tangent cones. Also, there are many different software tools available to do this. Instead of reinventing the wheel, `LattE` links with other software tools to compute these basic objects. In this section, we describe how you can control which software tool is used.

- The `4ti2` program can be used instead of `cddlib` to compute the vertex cones of polytopes, triangulations, and duals of cones. In many cases, `4ti2` is faster, which is why it is now the default when it is available.

```
--compute-vertex-cones={cdd,4ti2}
--triangulation={cddlib,4ti2}
--dualization={cdd,4ti2}
```

- By default, LattE assumes the h-representation may contain redundant hyperplanes and tries to find and remove them. You can control how much more LattE should spend checking the input h-representation with the following option.

  `--redundancy-check={none,cddlib,full-cddlib}.`

  - "full-cddlib" (the default) uses cddlib to compute an irredundant system of linear equations and inequalities describing the polyhedron. This corresponds to the traditional LattE behavior; it can be expensive.
  - "cddlib" (used to be the default in the 1.2+mk-0.9.x series) uses cddlib to compute some implicit linearities only; it often fails but is faster than full-cddlib.
  - "none" does nothing, the input description of the polytope should be irredundant.

3.4 How to use top-ehrhart-knapsack

To view the help menu, run

```
top-ehrhart-knapsack --help
```

An (equality-constrained) knapsack is a polytope in the form $\{x \in \mathbb{R}^d \mid \alpha_1 x_1 + \cdots + \alpha_d x_d = t; x_i \geq 0\}$, which we identify by the coefficients $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_d]$. `top-ehrhart-knapsack` can compute the largest k degree terms in the Ehrhart quasi-polynomial $E(\boldsymbol{\alpha}, t)$ as a function of t, using the algorithm from [2, 1]. We assume the greatest common divisor of the α_i is one, because if g is the greatest common divisor of the α_i and $g \neq 1$, then $E(\boldsymbol{\alpha}/g, t) = E(\boldsymbol{\alpha}, gt)$. There are only a few command line parameters:

–file, -f FILENAME FILENAME contains the knapsack file in a special format (see below). Required parameter.

–out, -o FILENAME2 Saves the result to a file. This is an optional parameter, but is is recommended to always save the result to a file as the output can be large.

-k n Computes the n^{th} term of the Ehrhart polynomial. This or "–all-k" is required.

–all-k n Computes the largest n^{th} terms of the Ehrhart polynomial.

–gcd-polynomial (0 | 1) If 1, uses a polynomial time algorithm to find the coefficients of the Ehrhart quasi-polynomial. The default is 1. See note below.

`top-ehrhart-knapsack` uses a special file format to input the knapsack coefficients. For a the knapsack $\boldsymbol{\alpha}$, the input file contains one line: the number of coefficients in the knapsack followed by each coefficient separated by spaces.

The two main computation options are best illustrated with an example. Consider the knapsack $\boldsymbol{\alpha} := [1, 2, 3, 4, 5]$. Notice that the greatest common divisor is one. Save this knapsack in a file called `knap1.txt`:

```
5 1 2 3 4 5
```

To compute the 2nd coefficient of the Ehrhart polynomial, use the command

```
top-ehrhart-knapsack --file knap1.txt -k 2 --out results.mpl
```

which gives the result

```
coeff4minus1:= (1/96);
```

and hence the 2nd term is $T^2/96$.

Next, we can compute the largest 5 terms (because the dimension of this knapsack is 4, this means we are computing the full Ehrhart polynomial), with the command

```
top-ehrhart-knapsack --file knap1.txt --all-k 5 --out results.mpl
```

which gives the result

```
coeff4minus0:= (1/2880);
coeff4minus1:= (((((-1/48) + (((((((1)*((MOD(t*(1/3),1))*(3))) + <and so on>;
coeff4minus2:= <another large expression>;
coeff4minus3:= <another large expression>;
coeff4minus4:= <another large expression>;

topKPolynomial:=(coeff4minus0)*T^(4) + (coeff4minus1)*T^(3)
 + (coeff4minus2)*T^(2) + (coeff4minus3)*T^(1) + (coeff4minus4)*T^(0);
```

There are a few things to note from this example:

- The expressions for each coefficient might be very long. It is recommended that you always save the results in an output file, and use a computer algebra system to simplify the expressions. Also note that the output can be parsed by `Maple`. In fact, using `Maple` to simplify the expressions, "coeff4minus1" is 1/96.

- The MOD(a,b) function represents the number in the half-open interval $[0, b)$ that is equal to $a \mod b$. For $b = 1$, $\mathrm{MOD}(a, 1)$ is the fractional part of a and is equal to $a - \lfloor a \rfloor \in [0, 1)$, where $\lfloor a \rfloor$ is the largest integer smaller or equal to a.
- The output uses "t" for the periodic coefficients, and "T" for the monomial terms. This is done so you can use `Maple`'s `coeff` command to extract the coefficients.
- To evaluate the polynomial, we can run

  ```
  eval(subs({T=10,t=10,MOD=latteMod}, topKPolynomial))
  ```

  where the function `latteMod` can be found in `code/maple/m-knapsack.mpl`.

The option `--gcd-polynomial` controls how poles are computed, which is given by the greatest common divisor of subsets of $\{\alpha_1, \ldots, \alpha_d\}$. For example, the options `--gcd-polynomial 1 -k 5` will require finding all $\binom{d}{5} + \binom{d}{4} + \cdots + \binom{d}{1}$ subsets of size at most 5 and compute the greatest common divisor of each one. However, `--gcd-polynomial 0 -k 5` will use a dynamic programming technique to compute the greatest common divisor of every subset of $\{\alpha_1, \ldots, \alpha_d\}$. If the number of unique greatest common divisors from subsets is much smaller than 2^d, and d is large, `--gcd-polynomial 0` should be much faster.

3.5 Optimization

`LattE` can also optimize over the integer points of a polytope. However, this part of the software is not as stable as the rest of the code. The optimization executables **require** a cost vector specified in "fileName.cost" if the polytope file is named "fileName."

- Maximizes/Minimizes a given linear cost function over the lattice points in the polytope. The Digging algorithm [8] is used. Optimal point and optimal value is returned.

  ```
  ./latte-maximize fileName
  ./latte-minimize fileName
  ```

- Maximizes/Minimizes a given linear cost function over the lattice points in the polytope. The Binary search algorithm is used. Only optimal value is returned.

  ```
  ./latte-maximize bbs fileName
  ./latte-minimize bbs fileName
  ```

4 Downloading and Installing LattE

LattE is downloadable from the following website:

http://www.math.ucdavis.edu/~latte/

LattE uses the GNU Autoconf, Automake, Libtool tools. The following README file provides detailed instructions.

```
This is LattE integrale, the official new version of LattE.

In addition to the traditional LattE function of counting lattice
points in polytopes by variants of Barvinok's algorithm, LattE
integrale can also compute volumes and integrate polynomial functions
over polytopes.  It supersedes LattE macchiato, an improved version of
LattE.

LattE requires the following programs and libraries:

  * GMP, compiled with --enable-cxx
  * NTL, version 5.4 or newer
  * cddlib
  * (optional) LRS
  * (optional) LiDIA
  * (optional) 4ti2
  * (optional) Maple (non-free)

If you do not have these libraries installed yet, follow the
instructions below to install them.  However, we also package a source
code distribution called

   "latte-integrale"

(also called LattE integrale "for tea, too") that includes all of
these libraries (except, of course, Maple) and will build them
automatically.  You can get it at the same place where you got this
package,

    http://www.math.ucdavis.edu/~latte/


Building and installing LattE
-----------------------------

It is STRONGLY RECOMMENDED to use the source code distribution called
"latte-integrale".  It contains all prerequisite libraries and also
PATCHES for some of the libraries that fix configuration and build
problems that are not yet included in upstream releases of the library.

If you do not wish to use "latte-integrale", follow the instructions
below.  The instructions assume you want to install LattE and all its
```

```
prerequisites into your home directory, namely into a hierarchy rooted
at the directory $HOME/latte.

1. Install the GNU Multiple Precision Library

   Download it from http://www.swox.com/gmp/
   Unpack it, then in the source directory do:

     ./configure --prefix=$HOME/latte --enable-cxx
     make
     make install

2. Install Victor Shoup's Number Theoretic Library

   Obtain it from http://www.shoup.net/ntl/
   Unpack it, then in the source directory do:

      cd src
      ./configure PREFIX=$HOME/latte GMP_PREFIX=$HOME/latte NTL_GMP_LIP=on
      make
      make install

3. Install Komei Fukuda's package cddlib

   Obtain then from http://www.ifor.math.ethz.ch/~fukuda/cdd_home

4. Put $HOME/latte/bin into your $PATH
   and $HOME/latte/lib into your $LD_LIBRARY_PATH:

      export PATH="$HOME/latte/bin:$PATH"
      export LD_LIBRARY_PATH="$HOME/latte/lib:$LD_LIBRARY_PATH"

5. Optionally, install the non-free library LiDIA.

   If you are using LiDIA 2.2.0, note that it installs the directory
   include/lidia but expects its header files in include/LiDIA.  We
   advise to put a symbolic link after installation of LiDIA.

6. Optionally, install 4ti2.

7. Optionally, if you have Maple, make sure that the directory where
   the command-line executable of Maple lives ("maple" or, on Windows,
   "cmaple.exe") is in your $PATH:

      export PATH="/path/to/maple/directory:$PATH"

N. Build and install LattE

   From the source directory of LattE:

      ./configure --prefix=$HOME/latte --with-default=$HOME/latte
```

```
      make
      make install

Now the LattE executables (count, integrate, latte-minimize, latte-maximize, ...)
should be available in $HOME/latte/bin.


More information
----------------

 * License: GNU General Public License, see COPYING

 * Authors: see AUTHORS

 * Documentation: See the LattE manual (file 'doc/manual.pdf') to get started.

 * Changes since the official release 1.2: see NEWS and ChangeLog

 * Website: http://www.math.ucdavis.edu/~latte
```

5 A Brief Tutorial

In this section we invite the reader to follow along a few examples that show how to use `LattE` and also how to counter-check results.

5.1 Counting Magic Squares

Our first example deals with counting magic 4×4 squares. We call a 4×4 array of nonnegative numbers a magic square if the sums of the 4 entries along each row, along each column and along the two main diagonals equals the same number s, the magic constant. Let us start with counting magic 4×4 squares that have the magic constant 1. Associating variables $x_1, \ldots, x_{16}$ with the 16 entries, the conditions of a magic 4×4 square of magic sum 1 can be encoded into the following input file "EXAMPLES/magic4x4" for `LattE`.

```
10 17
1 -1 -1 -1 -1  0  0  0  0  0  0  0  0  0  0  0  0
1  0  0  0  0 -1 -1 -1 -1  0  0  0  0  0  0  0  0
1  0  0  0  0  0  0  0  0 -1 -1 -1 -1  0  0  0  0
1  0  0  0  0  0  0  0  0  0  0  0  0 -1 -1 -1 -1
1 -1  0  0  0 -1  0  0  0 -1  0  0  0 -1  0  0  0
1  0 -1  0  0  0 -1  0  0  0 -1  0  0  0 -1  0  0
1  0  0 -1  0  0  0 -1  0  0  0 -1  0  0  0 -1  0
1  0  0  0 -1  0  0  0 -1  0  0  0 -1  0  0  0 -1
```

```
1 -1  0  0  0  0 -1  0  0  0  0 -1  0  0  0  0 -1
1  0  0  0 -1  0  0 -1  0  0 -1  0  0 -1  0  0  0
linearity 10 1 2 3 4 5 6 7 8 9 10
nonnegative 16 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

Now we simply invoke the counting function of `LattE` by typing:

```
count EXAMPLES/magic4x4
```

The last couple of lines that `LattE` prints to the screen look as follows:

```
Total Unimodular Cones: 418
Maximum number of simplicial cones in memory at once: 27

*****  Total number of lattice points: 8  ****

Computation done.
Time: 1.24219 sec
```

Therefore, there are exactly 8 magic 4×4 squares that have the magic constant 1. This is not yet impressive, as we could have done that by hand. Therefore, let us try and find the corresponding number for the magic constant 12. Since this problem is a dilation (by factor 12) of the original problem, we do not have to create a new file. Instead, we use the option "dilation" to indicate that we want to count the number of lattice points of a dilation of the given polytope:

```
count --dilation=12 EXAMPLES/magic4x4
```

The last couple of lines that `LattE` prints to the screen look as follows:

```
Total Unimodular Cones: 418
Maximum number of simplicial cones in memory at once: 27

*****  Total number of lattice points: 225351  ****

Computation done.
Time: 1.22656 sec
```

Therefore, there are exactly 225351 magic 4×4 squares that have the magic constant 12. (We would NOT want to do THAT one by hand, would we?!)

Here is some amazing observation: the running time of `LattE` is roughly the same for counting magic squares of sum 1 and of sum 12. This phenomenon is due to the fact that the main part of the computation, the creation of the generating function that encodes all lattice points in the polytope, is nearly identical in both cases.

Although we may be already happy with these simple counting results, let us be a bit more ambitious and and let us find a counting formula that, for given magic sum s, returns the number of magic 4×4 squares that have the magic constant s.

For this, simply type (note that `LattE` invokes `Maple` to simplify intermediate expressions):

```
count --simplified-ehrhart-series EXAMPLES/magic4x4
```

The last couple of lines that `LattE` prints to the screen looks as follows:

```
Rational function written to EXAMPLES/magic4x4.rat

Computation done.
Time: 0.724609 sec
```

We are informed that this call created a file "EXAMPLES/magic4x4.rat" containing the Ehrhart series as a rational function:

```
(t^8+4*t^7+18*t^6+36*t^5+50*t^4+36*t^3+18*t^2+4*t+1)/(-1+t)^4/(-1+t^2)^4
```

Now we could use `Maple` (or your favorite computer algebra software) to find a series expansion of this expression.

$$\begin{aligned} & \frac{t^8+4*t^7+18*t^6+36*t^5+50*t^4+36*t^3+18*t^2+4*t+1}{(-1+t)^4(-1+t^2)^4} \\ = \quad & 1+8t^1+48t^2+200t^3+675t^4+1904t^5+4736t^6+10608t^7+21925t^8+ \\ & 42328t^9+77328t^{10}+134680t^{11}+225351t^{12}+364000t^{13}+570368t^{14}+ \\ & 869856t^{15}+O(t^{16}) \end{aligned}$$

The summands $8t$ and $225351t^{12}$ reconfirm our previous counts.

Although this rational function encodes the full Ehrhart series, it is not always as easy to compute as for magic 4×4 squares. As it turns out, adding and simplifying rational functions, although in just one variable t, can be extremely costly due to the high powers in t and due to long integer coefficients that appear.

However, even if we cannot compute the full Ehrhart series, we can at least try and find the first couple of terms of it.

```
count --ehrhart-taylor=15 EXAMPLES/magic4x4
```

The last couple of lines that `LattE` prints to the screen look as follows:

```
Memory Save Mode: Taylor Expansion:
1
8t^1
48t^2
200t^3
675t^4
1904t^5
4736t^6
10608t^7
21925t^8
42328t^9
77328t^10
134680t^11
225351t^12
364000t^13
570368t^14
869856t^15
Computation done.
Time: 1.83789 sec
```

Again, our previous counts are reconfirmed.

Nice, but the more terms we want to compute the more time-consuming this task becomes. Clearly, if we could find sufficiently many terms, we could compute the full Ehrhart series expansion in terms of a rational function by interpolation.

5.2 Counting Lattice Points in the 24-Cell

Our next example deals with a well-known combinatorial object, the 24-cell. Its description is given in the file "EXAMPLES/24_cell":

```
24 5
2 -1  1 -1 -1
1  0  0 -1  0
2 -1  1 -1  1
2 -1  1  1  1
1  0  0  0  1
1  0  1  0  0
2  1 -1  1 -1
2  1  1 -1  1
2  1  1  1  1
1  1  0  0  0
2  1  1  1 -1
```

```
2  1  1 -1 -1
2  1 -1  1  1
2  1 -1 -1  1
2  1 -1 -1 -1
1  0  0  1  0
2 -1  1  1 -1
1  0  0  0 -1
2 -1 -1  1 -1
1  0 -1  0  0
2 -1 -1  1  1
2 -1 -1 -1  1
2 -1 -1 -1 -1
1 -1  0  0  0
```

Now we invoke the counting function of `LattE` by typing:

```
count EXAMPLES/24_cell
```

The last couple of lines that `LattE` prints to the screen look as follows:

```
Total Unimodular Cones: 240
Maximum number of simplicial cones in memory at once: 30

*****  Total number of lattice points: 33  ****

Computation done.
Time: 0.429686 sec
```

Therefore, there are exactly 33 lattice points in the 24-cell. We get the same result by using the homogenized Barvinok algorithm:

```
count --homog EXAMPLES/24_cell
```

The last couple of lines that `LattE` prints to the screen look as follows:

```
Memory Save Mode: Taylor Expansion:

****  Total number of lattice points is: 33  ****

Computation done.
Time: 0.957031 sec
```

5.3 Integrating over a polytope

Let us integrate the polynomial $w^2x^2y^4z^8 - 3/8x^2$ and the power of a linear form $3(w + 2x + 4y + 6z)^{10}$ over the 24-cell.

Create a file named "even.polynomial" that has on its first line the polynomial. See Section 2.6 for a review of the syntax.

```
[[1,[2,2,4,8]], [-3/8,[0,2,0,0]]]
```

After running the integration command using the triangulation method

```
integrate --valuation=integrate --triangulate --monomials=even.polynomial 24_cell
```

we see that the two monomials where decomposed into 406 powers of linear forms and the answer is

```
starting to integrate 406 linear forms.

Integration (using the triangulation method)
     Answer: -110535307/170059500
     Decimal: -0.649980195166985672661627253990522
     Time: 1.92 sec
Computational time (algorithms + processing + program control)
     Total time: 2.00 sec
```

Now create a new file named "power10.linearforms" that has in its first line the power of a linear form:

```
[[3,[10,[1,2,4,6]]]]
```

Then integrate this power of a linear form over the 24_cell using the cone decomposition method with the following command:

```
integrate --cone-decompose --linear-forms=power10.linearforms  24_cell
```

We see the answer is computed very quickly.

```
Integration (using the cone decomposition method)
     Answer: 59555515086/77
     Decimal: 773448247.87012987012987012987013
     Time: 0.02 sec
Computational time (algorithms + processing + program control)
     Total time: 0.07 sec
```

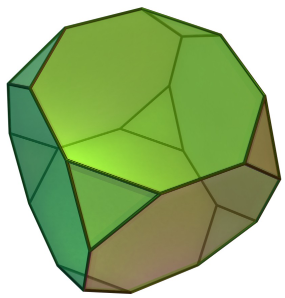


Figure 3: The truncated cube.

For the next example, consider the truncated cube in Figure 3

The vertices are

```
24 4
1 3 1 1
1 3 1 -1
1 3 -1 1
1 3 -1 -1
1 -3 1 1
1 -3 1 -1
1 -3 -1 1
1 -3 -1 -1
1 1 3 1
1 1 3 -1
1 1 -3 1
1 1 -3 -1
1 -1 3 1
1 -1 3 -1
1 -1 -3 1
1 -1 -3 -1
1 1 1 3
1 1 1 -3
1 1 -1 3
1 1 -1 -3
1 -1 1 3
1 -1 1 -3
1 -1 -1 3
```

```
1 -1 -1 -3
```

This time, let us enter the polynomial $x^{40}y^{40}z^{40}$ from stdin, which will be decomposed into $68,920$ powers of linear forms. Run

```
integrate --cone-decompose --triangulation=4ti2
          --vrep truncatedCube.vrep.latte
```

and type

```
p [[1,[40,40,40]]]
```

We see the exact answer is

$$\frac{9399128363294196571491924792863900251031820969229368882736399326510927664100376955325 6}{2795239135836124463932439643671211584534957465679791608181565}$$

This answer displays the power of using exact rational arithmetic!

5.4 Computing the Weighted Ehrhart Polynomial

Again, `LattE integrale` can (currently) only compute weighted Ehrhart polynomial functions if the user has `Maple`.

Consider the standard 0/1 square, which we represent in v-representation format as the file "square.vrep.latte":

```
4 3
1 0 0
1 1 0
1 0 1
1 1 1
```

Let's first compute the unweighted Ehrhart polynomial of the square:

```
integrate --valuation=top-ehrhart --vrep square.vrep.latte
```

and we get the answer $N^2 + 2N + 1$.

Note: For the case of the unweighted Ehrhart polynomial of a lattice polytope, when all coefficients are to be computed, it is faster to use `count --ehrhart-polynomial` instead. The benefit of `integrate --valuation=top-ehrhart` lies in the greater generality (polynomial weights, rational polytopes) and the ability to compute the top few coefficients, when the computation of the full Ehrhart polynomial is computationally intractable [4].

Next, compute the Ehrhart polynomial with weight $x_1^2 x_2^4$ where we enter the polynomial on the terminal:

```
integrate --valuation=top-ehrhart --interactive-mode
    --vrep square.vrep.latte
```

and then type

```
p
[[1,[2,4]]]
```

The answer is $1/15 * N^8 + 4/15 * N^7 + 71/180 * N^6 + 1/4 * N^5 + 2/45 * N^4 - 1/60 * N^3 - 1/180 * N^2$.

Finally, repeat the above commands, but find the polynomial that is correct if N is any rational or real dilation factor and save the answer to a file called "bigAnswer". That is, add `--real-dilations --top-ehrhart-save=bigAnswer` to the command line and use the same polynomial.

Open the file "bigAnswer." To evaluate the polynomial at a point $a \in \mathbb{R}$, first open the file "compute-top-ehrhart.mpl" that was made by `LattE integrale` and copy the same load path "compute-top-ehrhart.mpl" uses into "bigAnswer". Make your "bigAnswer" file look something like this (your file path might be different):

```
read("/home/latte/dest/share/latte-int/Conebyconeapproximations_08_11_2010.mpl"):

epoly:= ...;  # from bigAnswer
a := 5/2;
        # desired rational dilation factor

eval(subs({N=a,n=a, MOD=latteMod},epoly));
```

Note that you had to set both n and N to some dilation factor $a \in \mathbb{R}$ and the "latteMod" function is defined in the `Maple` script. If you then run this `Maple` file with the point $a = 5/2$ you will get 85.

5.5 Example of Optimization with `LattE`

Next, let us solve the problem "cuww1" [7, 8]. Its description is given in the file "EXAMPLES/cuww1":

```
1 6
89643482 -12223 -12224 -36674 -61119 -85569
linearity 1 1
nonnegative 5 1 2 3 4 5
```

The cost function can be found in the file "EXAMPLES/cuww1.cost":

```
1 5
213 -1928 -11111 -2345 9123
```

Now let us maximize this cost function over the given knapsack polytope. Note that by default, the digging algorithm as described in [8] is used.

```
./latte-maximize EXAMPLES/cuww1
```

The last couple of lines that `LattE` prints to the screen look as follows:

```
Finished computing a rational function.
Time: 0.158203 sec.

There is one optimal solution.

No digging.
An optimal solution for [213 -1928 -11111 -2345 9123] is: [7334 0 0 0 0].
The projected down opt value is: 191928257104
The optimal value is: 1562142.
The gap is: 7995261.806
Computation done.
Time: 0.203124 sec.
```

The solution $(7334, 0, 0, 0, 0)$ is quickly found. Now let us try to find the optimal value again by a different algorithm, the binary search algorithm.

```
./latte-maximize bbs EXAMPLES/cuww1
```

The last couple of lines that `LattE` prints to the screen look as follows:

```
Total of Iterations: 26
The total number of unimodular cones: 125562
The optimal value: 1562142

The number of optimal solutions: 1
Time: 0.042968
```

Note that we get the same optimal value, but no optimal solution is provided.

6 Release Information

6.1 System Requirements

`LattE` runs on Unix-like systems, including Mac OS X and GNU/Linux.

6.2 Additional `Maple` Connection

The call

```
count --simplified-ehrhart-series  fileName
```

requires `Maple` for simplifications of expressions. It should be sufficient to have a copy of `Maple` installed on your machine, without any additional special configuration required. `LattE` will still run even if `Maple` is not installed, but this simplification feature to "count" will not be available.

We have tested this connection with `Maple` 5.1, 8.0, and 14.0 and experienced no problem. Please let us know about any problem you experience with our connection to `Maple`.

6.3 License Agreement

This program is free software; you can redistribute it and/or modify it under the terms of the version 2 of GNU General Public License as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. (The careful user can find an install script for the source code in the directory "code/".)

You should have received a copy of the GNU General Public License along with this program; see the file COPYING. If not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

We have included a copy of the GNU General Public License also at the end of this document.

6.4 How to Cite LattE

Although LattE is free software, your acknowledgment is requested. If LattE is useful in your research or applications please acknowledge it by referencing this manual as

> V. Baldoni, N. Berline, J.A. De Loera, B. Dutra, M. Köppe, S. Moreinis, G. Pinto, M. Vergne, J. Wu,
> *A User's Guide for* LattE integrale *v1.7.2*, 2013, software package
> LattE is available at http://www.math.ucdavis.edu/~latte/

6.5 The LattE Team

Project directors

- Prof. Jesús A. De Loera (LattE v1.2, LattE integrale v1.5–)
- Prof. Matthias Köppe (LattE macchiato, LattE integrale v1.5–)

Students currently working on the project

- Brandon Dutra (LattE integrale v1.5–)

Distinguished LattE scientists, collaborators and advisors

- Prof. Raymond Hemmecke (LattE v1.2)
- Prof. Ruriko Yoshida (LattE v1.2)
- Dr. David Haws (LattE v1.2)
- Dr. Peter Huggins (LattE v1.2)
- Prof. Tyrrell McAllister
- Prof. Velleda Baldoni (LattE integrale v1.6)
- Prof. Nicole Berline (LattE integrale v1.6)
- Prof. Michele Vergne (LattE integrale v1.6)
- Prof. Alexander Barvinok
- Prof. Bernd Sturmfels

Alumni of the project

- Gregory Pinto (LattE integrale v1.5)
- Stanislav Moreinis (LattE integrale v1.5)
- Jianqiu Wu (LattE integrale v1.5)
- Jeremy Tauzer (LattE v1.2)
- Jonathan Brooks (LattE v1.2)
- Carol Shih (LattE v1.2)
- Esteban Pauli (LattE v1.2)
- Mike Zhang (LattE v1.2)

6.6 Acknowledgments

`LattE` currently uses many wonderful pieces of software. First is `cddlib` [12], developed by Komei Fukuda, whose webpage can be found at:

http://www.cs.mcgill.ca/∼fukuda/

Next, `LattE` uses `4ti2` [13] whose webpage can be found at:

http://www.4ti2.de

`cddlib` and `4ti2` is used for finding vertices of polytopes and the triangulation of cones.

In addition, `LattE` currently uses NTL, a Library for doing Number Theory, written by Victor Shoup [15], for LLL algorithm, matrix manipulations, storing variable length integers, and floating point numbers. NTL can be found at:

http://shoup.net/ntl/

In addition, `LattE` optionally uses the LiDIA library, developed at TU Darmstadt.

We are truly grateful to Velleda Baldoni, Alexander Barvinok, Nicole Berline, Komei Fukuda, Tyrrell McAllister, Dmitrii Pasechnik, Bernd Sturmfels, and Michèle Vergne for several suggestions and useful conversations that improved our software. We thank the National Science Foundation for support to this project via NSF grants DMS-0309694, DMS-0073815, DMS-0914107 and DMS-0914873. Most of the students were supported by those grants and by summer fellowships provided through the UC Davis VIGRE grants DMS-0135345 and DMS-0636297.

References

[1] V. Baldoni, N. Berline, J. A. De Loera, B. E. Dutra, M. Köppe, and M. Vergne, *Coefficients of Sylvester's denumerant*, eprint arXiv:1312.7147 [math.CO], 2013.

[2] ______, *Top degree coefficients of the denumerant*, 25th International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2013), DMTCS proc. AS, Discrete Mathematics and Theoretical Computer Science (DMTCS), 2013, pp. 1149–1160, available from `http://www.dmtcs.org/dmtcs-ojs/index.php/proceedings/article/view/dmAS0197/4324`.

[3] V. Baldoni, N. Berline, J. A. De Loera, M. Köppe, and M. Vergne, *How to integrate a polynomial over a simplex*, Mathematics of Computation **80** (2011), no. 273, 297–325, `doi:10.1090/S0025-5718-2010-02378-6`.

[4] ______, *Computation of the highest coefficients of weighted Ehrhart quasi-polynomials of rational polyhedra*, Foundations of Computational Mathematics **12** (2012), 435–469, `doi:10.1007/s10208-011-9106-4`.

[5] A. I. Barvinok, *Computing the Ehrhart quasi-polynomial of a rational simplex*, Math. Comp. **75** (2006), no. 255, 1449–1466.

[6] A. I. Barvinok and J. E. Pommersheim, *An algorithmic theory of lattice points in polyhedra*, New Perspectives in Algebraic Combinatorics (L. J. Billera, A. Björner, C. Greene, R. E. Simion, and R. P. Stanley, eds.), Math. Sci. Res. Inst. Publ., vol. 38, Cambridge Univ. Press, Cambridge, 1999, pp. 91–147.

[7] G. Cornuejols, R. Urbaniak, R. Weismantel, and L. Wolsey, *Decomposition of integer programs and of generating sets*, Algorithms ESA '97 (R. Burkard and G. Woeginger, eds.), Lecture Notes in Computer Science, vol. 1284, Springer Berlin / Heidelberg, 1997, 10.1007/3-540-63397-9_8, pp. 92–103, available from `http://dx.doi.org/10.1007/3-540-63397-9_8`.

[8] J. A. De Loera, D. Haws, R. Hemmecke, P. Huggins, and R. Yoshida, *Three kinds of integer programming algorithms based on barvinoks rational functions*, Integer Programming and Combinatorial Optimization (D. Bienstock and G. Nemhauser, eds.), Lecture Notes in Computer Science, vol. 3064, Springer Berlin / Heidelberg, 2004, 10.1007/978-3-540-25960-2_19, pp. 3–9, available from `http://dx.doi.org/10.1007/978-3-540-25960-2_19`.

[9] J. A. De Loera, B. Dutra, M. Köppe, S. Moreinis, G. Pinto, and J. Wu, *Software for exact integration of polynomials over polyhedra*, Computational Geometry: Theory and Applications **46** (2013), no. 3, 232–252, `doi:10.1016/j.comgeo.2012.09.001`.

[10] J. A. De Loera, R. Hemmecke, and M. Köppe, *Algebraic and geometric ideas in the theory of discrete optimization*, MOS–SIAM Series on Optimization, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2013, `doi:10.1137/1.9781611972443`, ISBN 978-1-61197-243-6.

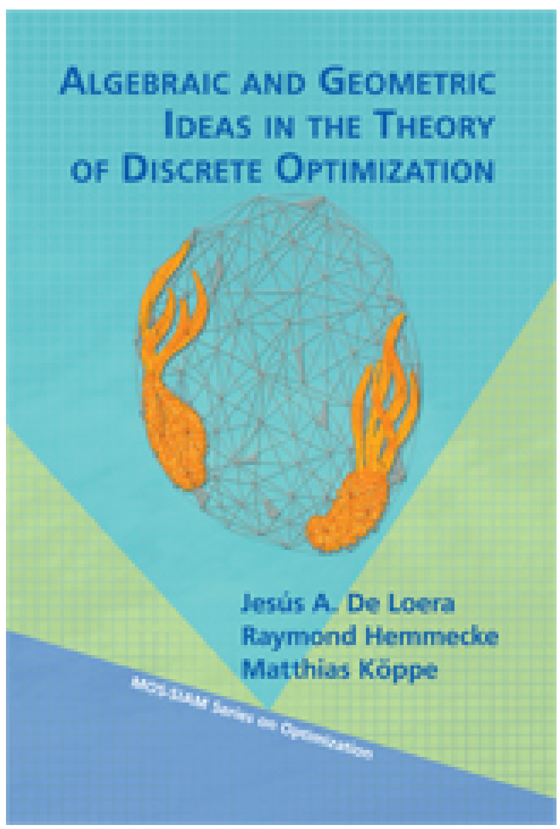
[11] J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida, *Effective lattice point counting in rational convex polytopes*, Journal of Symbolic Computation **38** (2004), no. 4, 1273 – 1302, Symbolic Computation in Algebra and Geometry, `doi:DOI:10.1016/j.jsc.2003.04.003`.

[12] K. Fukuda, *cddlib, version 094a*, Available from URL `http://www.cs.mcgill.ca/~fukuda/soft/cdd_home/cdd.html`, 2005.

[13] R. Hemmecke, R. Hemmecke, M. Köppe, P. Malkin, and M. Walter, *4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces*, Available at `http://www.4ti2.de`.

[14] M. Köppe, *A primal Barvinok algorithm based on irrational decompositions*, SIAM Journal on Discrete Mathematics **21** (2007), no. 1, 220–236, `doi:10.1137/060664768`.

[15] V. Shoup, *NTL, a library for doing number theory*, Available from URL `http://www.shoup.net/ntl/`, 2005.



A The GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

End of Terms and Conditions

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

> one line to give the program’s name and a brief idea of what it does.
> Copyright (C) yyyy name of author
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) yyyy name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
> This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program
> ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.
>
> signature of Ty Coon, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.