A User's Guide for `LattE integrale` v1.6 (Linux Release) *

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

January 2012

*Research supported by NSF grants DMS-0309694, NSF grants 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. © Department of Mathematics, University of California, Davis, 2011. This software is released under the GNU license agreement

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. LattE integrale was extended in 2012/2013 with a hybrid C++ and Maple implementation for computing the top coefficients of weighted Ehrhart polynomials. To learn more about the exact details of our implementation for lattice point enumeration, the interested reader can consult [7, 9] and the references listed therein. For learning the algorithmic details of integration, see [2, 8]. 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:

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

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 [3]) 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:

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

where

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

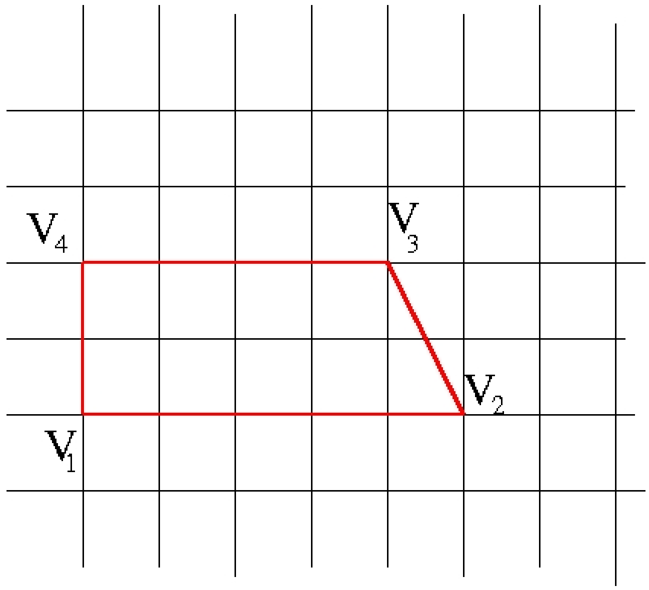

Figure 1: Quadrangle with vertices $V_1 = (0,0)$, $V_2 = (5,0)$, $V_3 = (4,2)$, and $V_4 = (0,2)$.

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

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

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 [7, 9] .

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 [2]. A decomposition of a polynomial as a sum of powers of linear forms is known as the polynomial Waring problem.

See [8] 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

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

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 $s_2 = (1,1)$:

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

Vertex $s_3 = (1,0)$:

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

Vertex $s_4 = (0,1)$:

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

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 Polynomials

`LattE integrale` can also compute the weighted Ehrhart polynomials where the weight function is a polynomial or a sum of powers of linear forms. However, the polytope must be a simplicial polytope and `Maple` must be installed on your computer.

1.2 What can `LattE` compute?

`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 polynomials.

max/minimize perform linear integer optimization.

The other executables in latte are drivers, converters, and other small utility function described in Section **??**.

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.

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 index of constraint, start counting from 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 short-hand.

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 index, start counting from 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
```

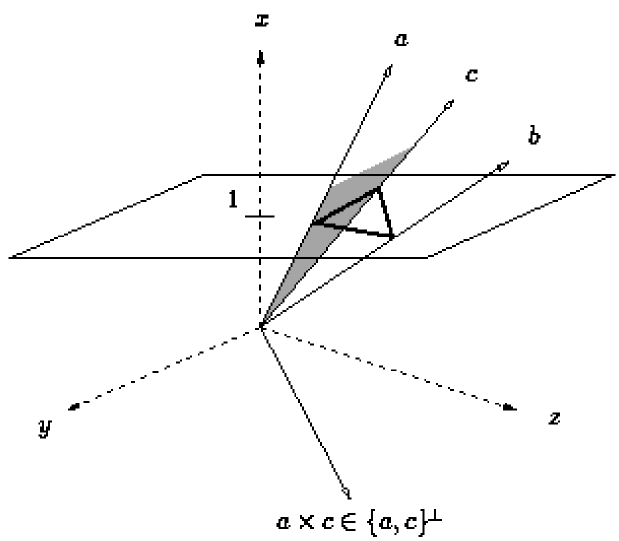

Figure 2: Homogenized triangule.

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:

$k\ (n+1)$
$1\ v_1$
$\vdots$
$1\ v_k$

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 [6].

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 count and volume of the input polytope. The current `LattE integrale` cannot integrate 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 used 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. We recommend *.vrep.latte and *.hrep.latte for LattE style files, but you are free to name your files anything.
2. CDD also requires "H-representation" or "V-representation" keywords in the file. Forgetting about the "linearity" and "nonnegative" keywords, there is no difference between a LattE v- and h-representation file.

2.6 Polynomials and linear forms

`LattE integrale` can also integrate polynomials and in particular sums of powers of linear forms. 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

  `[`$linear\text{-}term_1$`, `$linear\text{-}term_2$`, ...]`,

  where $linear\text{-}term_i$ 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 [2].

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`

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 [5] 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 `--compute-vertex-cones=4ti2`, as other methods in `LattE` currently refuse to handle unbounded polyhedra. For example,

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

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

- Compute the Ehrhart polynomial of an *integral* polytope

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

- 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 [7].

- 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
```

  This gives a new method for computing Ehrhart polynomials of integral polytopes in the primal space

```
./count --all-primal --ehrhart-polynomial 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 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 cal 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 weighted Ehrhart polynomials

LattE integrale can (currently) only compute weighted Ehrhart polynomials if the polytope is simple and if the user has Maple.

- Sets what you want to compute: weighted ehrhart polynomials.

```
--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 polynomial'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 polynomial's coefficients are computed incrementally which takes more memory but you may manually stop the computation at any time.

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

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

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

- Compute the weighted Ehrhart polynomial that is valid for non-integer dilations.

```
--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 polynomial 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 polynomial that is valid for real dilations.

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

- Compute weighted Ehrhart polynomial 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 polynomial

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

- Manually enter a weight function and save the polynomial to a file

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

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` and `CDD+` to compute the vertex cones of polytopes, triangulations, and duals of cones. In many cases, `4ti2` is faster.

```
--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 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 [5] 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 and Automake tools. Please see the README file in the LattE directory for detailed directions for installing 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.

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

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: -0.649980195166985672661627253990 52
     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: 773448247.870129870129870129870 13
     Time: 0.02 sec
Computational time (algorithms + processing + program control)
     Total time: 0.07 sec
```

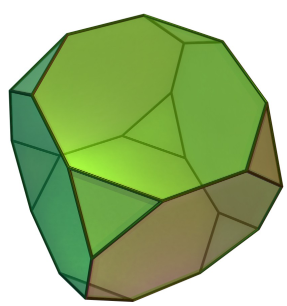

Figure 3: The truncated cube.

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

$$\frac{93991283632941965714919247928639002510318209692293688827363993265109276641003769553256}{27952391358361244639324396436712115845349574656797916081815 65}$$

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 polytope is simple and the user has `Maple`.

Consider a simple cube with vertices

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

Lets first compute the unweighted Ehrhart polynomial

```
./integrate --valuation=top-ehrhart --vrep cube.vrep.latte
```

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

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

```
./integrate --valuation=top-ehrhart --interactive-mode --vrep cube.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 a rational dilation 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:= ‘‘something big’’;

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

Note that you had to set both n and N to the point $a \in \mathbb{R}$ and the "latteMod" function is defined in the `Maple` script. If you then run this `Maple` file with the point $a = 2.5$ you will get 85.00000031.

5.5 Example of Optimization with LattE

Next, let us solve the problem "cuww1" [4, 5]. 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 [5] 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 and 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

De Loera, J.A., Dutra, B., Köppe, M., Moreinis, S., Pinto, G., Wu, J.
A User's Guide for LattE integrale *v1.6*, 2011, 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 cdd [6], developed by Komei Fukuda, whose webpage can be found at:

http://www.cs.mcgill.ca/~fukuda/

Next, LattE uses 4ti2 [1] whose webpage can be found at:

http://www.4ti2.de

cdd 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 [10], for LLL algorithm, matrix manipulations, storing variable length integers, and floating point numbers. NTL can be found at:

http://shoup.net/ntl/

We are truly grateful to Velleda Baldoni, Alexander Barvinok, Nicole Berline, Komei Fukuda, Tyrrell McAllister, Dmitrii Pasechnik, Michele Vergne, and Bernd Sturmfels 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] 4ti2 team, *4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces*, Available at www.4ti2.de.

[2] 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.

[3] 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.

[4] 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.

[5] J. 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.

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

[7] M. Köppe, *A primal Barvinok algorithm based on irrational decompositions*, SIAM Journal on Discrete Mathematics **21** (2007), no. 1, 220–236.

[8] J. A. D. Loera, B. Dutra, M. Köppe, S. Moreinis, G. Pinto, and J. Wu, *Software for exact integration of polynomials over polyhedra*, Manuscript, 2011.

[9] J. A. D. 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.

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

7 The GNU General Public License

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.

59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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.