cddlib Reference Manual

Komei Fukuda
Institute for Operations Research
and Institute of Theoretical Computer Science
ETH Zentrum, CH-8092 Zurich, Switzerland
and
Mathematics Institute / ROSO
EPFL, CH-1015 Lausanne, Switzerland
email: fukuda@ifor.math.ethz.ch

(cddlib ver. 0.94, manual ver. August 24, 2005)

Contents

Abstract

This is a reference manual for cddlib-094. The manual describes the library functions and data types implemented in the cddlib C-library which is to perform fundamental polyhedral computations such as representation conversions and linear programming in both floating-point and GMP rational exact arithmetic. Please read the accompanying README file and test programs to complement the manual.

The new functions added in this version include `dd_MatrixCanonicalize` to find a non-redundant proper H- or V-representation, `dd_FindRelativeInterior` to find a relative interior point of an H-polyhedron, and `dd_ExistsRestrictedFace` (Farkas-type alternative theorem verifier) to check the existence of a point satisfying a specified system of linear inequalities possibly including multiple strict inequalities.

The new functions are particularly important for the development of related software packages MinkSum (by Ch. Weibel) and Gfan (by Anders Jensen),

1 Introduction

The program cddlib is an efficient implementation [16] of the double description Method [19] for generating all vertices (i.e. extreme points) and extreme rays of a general convex polyhedron given by a system of linear inequalities:

$$P = \{x = (x_1, x_2, \ldots, x_d)^T \in R^d : b - Ax \geq 0\}$$

where A is a given $m \times d$ real matrix and b is a given real m-vector. In the mathematical language, the computation is the transformation of an *H-representation* of a convex polytope to an *V-representation.*

cddlib is a C-library version of the previously released C-code cdd/cdd+. In order to make this library version, a large part of the cdd source (Version 0.61) has been rewritten. This library version is more flexible since it can be called from other programs in C/C++. Unlike cdd/cdd+, cddlib can handle any general input and is more general. Furtthermore, additional functions have been written to extend its functionality.

One useful feature of cddlib/cdd/cdd+ is its capability of handling the dual (reverse) problem without any transformation of data. The dual transformation problem of a V-representation to a minimal H-representation and is often called the *(convex) hull problem.* More explicitly, is to obtain a linear inequality representation of a convex polyhedron given as the Minkowski sum of the convex hull of a finite set of points and the nonnegative hull of a finite set of points in R^d:

$$P = conv(v_1, \ldots, v_n) + nonneg(r_{n+1}, \ldots, r_{n+s}),$$

where the *Minkowski sum of two subsets S and T* of R^d is defined as

$$S + T = \{s + t \mid s \in S \text{ and } t \in T\}.$$

As we see in this manual, the computation can be done in straightforward manner. Unlike the earlier versions of cdd/cdd+ that assume certain regularity conditions for input, cddlib is designed to do a correct transformation for any general input. The user must be aware of the fact that in certain cases the transformation is not unique and there are polyhedra with infinitely many representations. For example, a line segment (1-dimensional polytope) in R^3 has infinitely many minimal H-representations, and a halfspace in the same space has infinitely many minimal V-representations. cddlib generates merely one minimal representation.

cddlib comes with an LP code to solve the general linear programming (LP) problem to maximize (or minimize) a linear function over polyhedron P. It is useful mainly for solving dense LP's with large m (say, up to few hundred thousands) and small d (say, up to 100). It implements a revised dual simplex method that updates $(d+1)\times(d+1)$ matrix for a pivot operation.

The program cddlib has an I/O routines that read and write files in *Polyhedra format* which was defined by David Avis and the author in 1993, and has been updated in 1997 and 1999. The program called lrs and lrslib [2] developed by David Avis is a C-implementation of the reverse search algorithm [4] for the same enumeration purpose, and it conforms to Polyhedra format as well. Hopefully, this compatibility of the two programs enables users to use both programs for the same input files and to choose whichever is useful for their purposes. From our experiences with relatively large problems, the two methods are both useful and perhaps complementary to each other. In general, the program cddlib tends to be efficient for highly degenerate inputs and the program rs tends to be efficient for nondegenerate or slightly degenerate problems.

Although the program can be used for nondegenerate inputs, it might not be very efficient. For nondegenerate inputs, other available programs, such as the reverse search code lrs or qhull (developed by the Geometry Center), might be more efficient. See Section 8 for pointers to these codes. The paper [3] contains many interesting results on polyhedral computation and experimental results on cdd+, lrs, qhull and porta.

This program can be distributed freely under the GNU GENERAL PUBLIC LICENSE. Please read the file COPYING carefully before using.

I will not take any responsibility of any problems you might have with this program. But I will be glad to receive bug reports or suggestions at the e-mail addresses above. If cddlib turns out to be useful, please kindly inform me of what purposes cdd has been used for. I will be happy to include a list of applications in future distribution if I receive enough replies. The most powerful support for free software development is user's appreciation and collaboration.

2 Polyhedra H- and V-Formats (Version 1999)

Every convex polyhedron has two representations, one as the intersection of finite halfspaces and the other as Minkowski sum of the convex hull of finite points and the nonnegative hull of finite directions. These are called H-representation and V-representation, respectively.

Naturally there are two basic Polyhedra formats, H-format for H-representation and V-format for V-representation. These two formats are designed to be almost indistinguishable, and in fact, one can almost pretend one for the other. There is some asymmetry arising from the asymmetry of two representations.

First we start with the H-representation. Let A be an $m\times d$ matrix, and let b be a column m-vector. The Polyhedra format (*H-format*) of the system $b - Ax \geq \mathbf{0}$ of m inequalities in d variables $x = (x_1, x_2, \ldots, x_d)^T$ is

various comments
H-representation
(**linearity** t i_1 i_2 $\ldots$ i_t)
begin
m $d+1$ numbertype
b $-A$
end
various options

where numbertype can be one of integer, rational or real. When rational type is selected, each component of b and A can be specified by the usual integer expression or by the rational expression "p/q" or "$-p/q$" where p and q are arbitrary long positive integers (see the example input file rational.ine). In the 1997 format, we introduced "H-representation" which must appear before "begin". There was one restriction in the old polyhedra format (before 1997): the last d rows must determine a vertex of P. This is obsolete now.

In the new 1999 format, we added the possibility of specifying **linearity**. This means that for H-representation, some of the input rows can be specified as **equalities**: $b_{i_j} - A_{i_j}x = 0$ for all $j = 1, 2, \ldots, t$. The linearity line may be omitted if there are no equalities.

Option lines can be used to control computation of a specific program. In particular both cdd and lrs use the option lines to represent a linear objective function. See the attached LP files, samplelp*.ine.

Next we define Polyhedra *V-format*. Let P be represented by n gerating points and s generating directions (rays) as $P = conv(v_1, \ldots, v_n) + nonneg(r_{n+1}, \ldots, r_{n+s})$. Then the Polyhedra V-format for P is

various comments		
V-representation		
(**linearity** t i_1 i_2 $\ldots$ i_t)		
begin		
$n+s$	$d+1$	numbertype
1	v_1	
$\vdots$	$\vdots$	
1	v_n	
0	r_{n+1}	
$\vdots$	$\vdots$	
0	r_{n+s}	
end		
various options		

Here we do not require that vertices and rays are listed separately; they can appear mixed in arbitrary order.

Linearity for V-representation specifies a subset of generators whose coefficients are relaxed to be **free**: for all $j = 1, 2, \ldots, t$, the $k = i_j$th generator (v_k or r_k whichever is the i_jth generator) is a free generator. This means for each such a ray r_k, the line generated by r_k is in the polyhedron, and for each such a vertex v_k, its coefficient is no longer nonnegative but still the coefficients for all v_i's must sum up to one. It is highly unlikely that one needs to use linearity for vertex generators, and it is defined mostly for formality.

When the representation statement, either "H-representation" or "V-representation", is omitted, the former "H-representation" is assumed.

It is strongly suggested to use the following rule for naming H-format files and V-format files:

(a) use the filename extension ".ine" for H-files (where ine stands for inequalities), and

(b) use the filename extension ".ext" for V-files (where ext stands for extreme points/rays).

3 Basic Object Types (Structures) in cddlib

Here are the types (defined in cddtypes.h) that are important for the cddlib user. The most important one, `dd_MatrixType`, is to store a Polyhedra data in a straightforward manner. Once the user sets up a (pointer to) `dd_MatrixType` data, he/she can load the data to an internal data type (`dd_PolyhedraType`) by using functions described in the next section, and apply the double descrition method to get another representation. As an option `dd_MatrixType` can save a linear objective function to be used by a linear programming solver.

The two dimensional array data in the structure `dd_MatrixType` is `dd_Amatrix` whose components are of type `mytype`. The type mytype is set to be either the rational type `mpq_t` of the GNU MP Library or the C double array of size 1. This abstract type allows us to write a single program that can be compiled with the two or more different arithmetics, see example programs such as simplecdd.c, testlp*.c and testcdd*.c in the `src` and `src-gmp` subdirectories of the source distribution.

There is another data type that is used very often, `dd_SetFamilyType`. This is to store a family of subsets of a finite set. Such a family can represent the incidence relations between the set of extreme points and the set of facets of a polyhedron. Also, it can represent a graph structure by listing the set of vertices adjacent to each vertex (i.e. the adjacency list). To implement `dd_SetFamilyType`, we use a separate set library called `setoper`, that handles the basic set operations, This library is briefly introduced in Section 4.6.

```
#define dd_FALSE 0
#define dd_TRUE 1

typedef long dd_rowrange;
typedef long dd_colrange;
typedef long dd_bigrange;

typedef set_type dd_rowset;   /* set_type defined in setoper.h */
typedef set_type dd_colset;
typedef long *dd_rowindex;
typedef int *dd_rowflag;
typedef long *dd_colindex;
typedef mytype **dd_Amatrix;  /* mytype is either GMP mpq_t or 1-dim double array. */
typedef mytype *dd_Arow;
typedef set_type *dd_SetVector;

typedef enum {
  dd_Real, dd_Rational, dd_Integer, dd_Unknown
} dd_NumberType;

typedef enum {
  dd_Inequality, dd_Generator, dd_Unspecified
} dd_RepresentationType;

typedef enum {
  dd_MaxIndex, dd_MinIndex, dd_MinCutoff, dd_MaxCutoff, dd_MixCutoff,
   dd_LexMin, dd_LexMax, dd_RandomRow
```

```
} dd_RowOrderType;

typedef enum {
  dd_InProgress, dd_AllFound, dd_RegionEmpty
} dd_CompStatusType;

typedef enum {
  dd_DimensionTooLarge, dd_ImproperInputFormat,
  dd_NegativeMatrixSize, dd_EmptyVrepresentation,
  dd_IFileNotFound, dd_OFileNotOpen, dd_NoLPObjective,
  dd_NoRealNumberSupport, dd_NoError
} dd_ErrorType;

typedef enum {
  dd_LPnone=0, dd_LPmax, dd_LPmin
} dd_LPObjectiveType;

typedef enum {
  dd_LPSundecided, dd_Optimal, dd_Inconsistent, dd_DualInconsistent,
  dd_StrucInconsistent, dd_StrucDualInconsistent,
  dd_Unbounded, dd_DualUnbounded
} dd_LPStatusType;

typedef struct matrixdata *dd_MatrixPtr;
typedef struct matrixdata {
  dd_rowrange rowsize;
  dd_rowset linset;
    /*  a subset of rows of linearity (ie, generators of
        linearity space for V-representation, and equations
        for H-representation. */
  dd_colrange colsize;
  dd_RepresentationType representation;
  dd_NumberType numbtype;
  dd_Amatrix matrix;
  dd_LPObjectiveType objective;
  dd_Arow rowvec;
}  dd_MatrixType;

typedef struct setfamily *dd_SetFamilyPtr;
typedef struct setfamily {
  dd_bigrange famsize;
  dd_bigrange setsize;
  dd_SetVector set;
} dd_SetFamilyType;

typedef struct lpsolution *dd_LPSolutionPtr;
typedef struct lpsolution {
  dd_DataFileType filename;
```

```
  dd_LPObjectiveType objective;
  dd_LPSolverType solver;
  dd_rowrange m;
  dd_colrange d;
  dd_NumberType numbtype;

  dd_LPStatusType LPS;  /* the current solution status */
  mytype optvalue;  /* optimal value */
  dd_Arow sol;   /* primal solution */
  dd_Arow dsol;  /* dual solution */
  dd_colindex nbindex;  /* current basis represented by nonbasic indices */
  dd_rowrange re;  /* row index as a certificate in the case of inconsistency */
  dd_colrange se;  /* col index as a certificate in the case of dual inconsistency */
  long pivots[5];
   /* pivots[0]=setup (to find a basis), pivots[1]=PhaseI or Criss-Cross,
      pivots[2]=Phase II, pivots[3]=Anticycling, pivots[4]=GMP postopt  */
  long total_pivots;
} dd_LPSolutionType;
```

4 Library Functions

Here we list some of the most important library functions/procedures. We use the following convention: `poly` is of type `dd_PolyhedraPtr`, `matrix`, `matrix1` and `matrix2` are of type `dd_MatrixPtr`, `matrixP`, of type `dd_MatrixPtr*`, `err` is of type `dd_ErrorType*`, `ifile` and `ofile` are of type `char*`, `A` is of type `dd_Amatrix`, `point` and `vector` are of type `dd_Arow`, `d` is of type `dd_colrange`, `m` and `i` are of type `dd_rowrange`, `x` is of type `mytype`, `a` is of type `signed long integer`, `b` is of type `double`, `set` is of type `set_type`. Also, `setfam` is of type `dd_SetFamilyPtr`, `lp` is of type `dd_LPPtr`, `lps` is of type `dd_LPSolutionPtr`, `solver` is of type `dd_LPSolverType`, `roworder` is of type `dd_RowOrderType`.

4.1 Library Initialization

`void dd_set_global_constants(void) :`

This is to set the global constants such as `dd_zero`, `dd_purezero` and `dd_one` for sign recognition and basic arithmetic operations. Every program to use cddlib must call this function before doing any computation. Just call this once. See Section 4.3.3 for the definitions of constants.

4.2 Core Functions

There are two types of core functions in cddlib. The first type runs the double description (DD) algorithm and does a representation conversion of a specified polyhedron. The standard header for this type is `dd_DD*`. The second type solves one or more linear programs with no special headers. Both types of computations are nontrivial and the users (especially for the DD algorithm) must know that there is a serous limit in the sizes of problems that can be practically solved. Please check *.ext and *.ine files that come with cddlib to get ideas of tractable problems.

In addition to previously defined objects, the symbol `roworder` is of `dd_RowOrderType`. The symbol `matrixP` is a pointer to **dd_MatrixType**. the arguments `impl_lin` and `redset` are both a pointer to `dd_rowset` type, and `newpos` is a pointer to `dd_rowindex` type.

`dd_PolyhedraPtr dd_DDMatrix2Poly(matrix, err) :`
: Store the representation given by `matrix` in a polyhedra data, and generate the second representation of `*poly`. It returns a pointer to the data. `*err` returns `dd_NoError` if the computation terminates normally. Otherwise, it returns a value according to the error occured.

`dd_PolyhedraPtr dd_DDMatrix2Poly2(matrix, roworder, err) :`
: This is the same function as `dd_DDMatrix2Poly` except that the insertion order is specified by the user. The argument `roworder` is of `dd_RowOrderType` and takes one of the values: `dd_MaxIndex`, `dd_MinIndex`, `dd_MinCutoff`, `dd_MaxCutoff`, `dd_MixCutoff`, `dd_LexMin`, `dd_LexMax`, `dd_RandomRow`. In general, `dd_LexMin` is the best choice which is in fact chosen in `dd_DDMatrix2Poly`. If you know that the input is already sorted in the order you like, use `dd_MinIndex` or `dd_MaxIndex`. If the input contains many redundant rows (say more than 80% redundant), you might want to try `dd_MaxCutoff` which might result in much faster termination, see [3, 16]

`boolean dd_DDInputAppend(poly, matrix, err) :`
: Modify the input representation in `*poly` by appending the matrix of `*matrix`, and compute the second representation. The number of columns in `*matrix` must be equal to the input representation.

`boolean dd_LPSolve(lp, solver, err) :`
: Solve `lp` by the algorithm `solver` and save the solututions in `*lp`. Unlike the earlier versions (dplex, cdd+), it can deal with equations and totally zero right hand sides. It is recommended that `solver` is `dd_DualSimplex`, the revised dual simplex method that updates a $d \times d$ dual basis matrix in each pivot (where d is the column size of lp).

  The revised dual simplex method is ideal for dense LPs in small number of variables (i.e. small column size, typically less than 100) and many inequality constraints (i.e. large row size, can be a few ten thousands). If your LP has many variables but only few constraints, solve the dual LP by this function.

  When it is compiled for GMP rational arithmetic, it first tries to solve an LP with C double floating-point arithmetic and verifies whether the output basis is correct with GMP. If so, the correct solution is computed with GMP. Otherwise, it (re)solves the LP from scratch with GMP. This is newly implemented in the version 093. The original (non-crossover) version of the same function is still available as `boolean dd_LPSolve0`.

`dd_boolean dd_Redundant(matrix, i, point, err) :`
: Check whether ith data in `matrix` is redundant for the representation. If it is nonredundant, it returns a certificate. For H-representation, it is a `point` in R^d which satisfies all inequalities except for the ith inequality. If i is a linearity, it does nothing and always returns `dd_FALSE`.

`dd_rowset dd_RedundantRows(matrix, err) :`
: Returns a maximal set of row indices such that the associated rows can be eliminated without changing the polyhedron. The function works for both V- and H-representations.

`dd_boolean dd_SRedundant(matrix, i, point, err) :`
: Check whether ith data in `matrix` is strongly redundant for the representation. If i is a linearity, it does nothing and always returns `dd_FALSE`. Here, ith inequality in H-representation

is *strongly redundant* if it is redundant and there is no point in the polyhedron satisfying the inequality with equality. In V-representation, ith point is *strongly redundant* if it is redundant and it is in the relative interior of the polyhedron. If it is not strongly redundant, it returns a certificate.

`dd_boolean dd_ImplicitLinearity(matrix, i, err) :`
Check whether ith row in the input is forced to be linearity (equality for H-representation). If i is linearity itself, it does nothing and always returns `dd_FALSE`.

`dd_rowset dd_ImplicitLinearityRows(matrix, err) :`
Returns the set of indices of rows that are implicitly linearity. It simply calls the library function `dd_ImplicitLinearity` for each inequality and collects the row indices for which the answer is `dd_TRUE`.

`dd_boolean dd_MatrixCanonicalize(matrixP, impl_lin, redset, newpos, err) :`
The input is a pointer `matrixP` to a matrix and the function modifies the matrix by putting a maximally linear independent linearities (basis) at the top of the matrix, and removing all redundant data. All implicit linearities and all (removed) redundant rows in the original matrix will be returned in the corresponding row sets. The new positions of the original rows are returned by the array `newpos`.

The cardinality of the new linearity set `(*matrixP)->linset` is the codimension of the polyhedron if it is H-polyhedron, and is the dimension of linearity space if it is V-polyhedron.

Note that the present version should not be called a canonicalization because it may generate two different representations of the same polyhedron. In the future, this function is expected to be correctly implemented.

`dd_boolean dd_MatrixCanonicalizeLinearity(matrixP, impl_linset, newpos. err) :`
It does only the first half of `dd_boolean dd_MatrixCanonicalize`, namely, it detects all implicit linearities and puts a maximally independent linearities at the top of the matrix. For example, this function can be used to detect the dimension of an H-polyhedron.

`dd_boolean dd_MatrixRedundancyRemove(matrixP, redset, newpos, err) :`
It does essentially the second half of `dd_boolean dd_MatrixCanonicalize`, namely, it detects all redundancies. This function should be used after `dd_MatrixCanonicalizeLinearity` has been called.

`dd_boolean dd_FindRelativeInterior(matrix, impl_lin, lin_basis, lps, err) :`
Computes a point in the relative interior of an H-polyhedron given by matrix, by solving an LP. The point will be returned by `lps`. See the sample program allfaces.c that generates all nonempty faces of an H-polyhedron and a relative interior point for each face. The former returns all implicit linearity rows (implicit equations) and the latter returns a basis of the union of linearity rows and implicit linearity rows. This means that the cardinality of `*lin_basis` is the codimension of the polyhedron.

`dd_boolean dd_ExistsRestrictedFace(matrix, R, S, err) :`
Returns the answer to the Farkas' type decision problem as to whether there is a point in the polyhedron given by matrix satisfying all constraints in `R` with equality and all constraints in `S` with strict inequality. More precisely, it is the linear feasibility problem:

$$\begin{array}{lllll} \exists ? & x & \text{satisfying} & b_r - A_r x & = 0,\ \forall r \in R \cup L \\ & & & b_s - A_s x & > 0,\ \forall s \in S \\ & & & b_t - A_t x & \geq 0,\ \forall t \in T, \end{array}$$

where L is the set of linearity rows of `matrix`, and T represents the set of rows that are not in $R \cup L \cup S$. Both `R` and `S` are of `dd_rowset` type. The set S is supposed to be disjoint from both R and L. If it is not the case, the set S will be considered as $S \setminus (R \cup L)$.

This function ignores `matrix->representation`, and thus even if it is set to `dd_Generator` or `dd_Unspecified`, it treats the matrix as if it were inequality representation.

`dd_boolean dd_ExistsRestrictedFace2(matrix, R, S, lps, err)` :
It is the same as the function `dd_ExistsRestrictedFace` except that it returns also a certificate for the answer. The certificate is a solution to the bounded LP:

$$
\begin{array}{llllll}
\text{(P)} & \max z & \text{subject to} & b_r - A_r x & & = 0, \ \forall r \in R \cup L \\
 & & & b_s - A_s x & -z & \geq 0, \ \forall s \in S \\
 & & & b_t - A_t x & & \geq 0, \ \forall t \in T \\
 & & & 1 & -z & \geq 0,
\end{array}
$$

where L is the set of linearity rows of `matrix`, and T represents the set of rows that are not in $R \cup L \cup S$. The answer for the decision problem is YES if and only if the LP attains an optimal and the optimal value is positive. The dual solution (either an optimal solution or a dual unbounded direction) can be considered as a certificate for the NO answer, if the answer is NO.

This function ignores `matrix->representation`, and thus even if it is set to `dd_Generator` or `dd_Unspecified`, it treats the matrix as if it were inequality representation.

`dd_SetFamilyPtr dd_Matrix2Adjacency(matrix, err)` :
Computes the adjacency list of input rows using the LP solver and without running the representation conversion. When the input is H-representation, it gives the facet graph of the polyhedron. For V-representation, it gives the (vertex) graph of the polyhedron. It is required that the input matrix is a minimal representation. Run redundancy removal functions before calling this function, see the sample code adjacency.c.

`dd_SetFamilyPtr dd_Matrix2WeakAdjacency(matrix, err)` :
Computes the weak adjacency list of input rows using the LP solver and without running the representation conversion. When the input is H-representation, it gives the graph where its nodes are the facets two nodes are adjacent if and only if the associated facets have some intersection. For V-representation, it gives the graph where its nodes are the vertices and two nodes are adjacent if and only if the associated vertices are on a common facet. It is required that the input matrix is a minimal representation. Run redundancy removal functions before calling this function, see the sample code adjacency.c.

`dd_MatrixPtr dd_FourierElimination(matrix, err)` :
Eliminate the last variable from a system of linear inequalities given by matrix by using the Fourier's Elimination. If the input matrix is V-representation, `*err` returns `dd_NotAvailForV`. This function does not remove redundancy and one might want to call redundancy removal functions afterwards. See the sample code fourier.c.

`dd_MatrixPtr dd_BlockElimination(matrix, set, err)` :
Eliminate a set of variables from a system of linear inequalities given by matrix by using the extreme rays of the dual linear system. See comments in the code cddproj.c for details. This might be a faster way to eliminate variables than the repeated FourierElimination when the number of variables to eliminate is large. If the input matrix is V-representation, `*err`

returns `dd_NotAvailForV`. This function does not remove redundancy and one might want to call redundancy removal functions afterwards. See the sample code projection.c.

`dd_rowrange dd_RayShooting(matrix, point, vector) :`
Finds the index of a halfspace first left by the ray starting from `point` toward the direction `vector`. It resolves tie by a lexicographic perturbation. Those inequalities violated by `point` will be simply ignored.

4.3 Data Manipulations

4.3.1 Number Assignments

For number assignments, one cannot use such expressions as `x=(mytype)a`. This is because cddlib uses an abstract number type (`mytype`) so that it can compute with various number types such as C double and GMP rational. User can easily add a new number type by redefining arithmetic operations in cddmp.h and cddmp.c.

`void dd_init(x) :`
This is to initialize a `mytype` variable `x` and to set it to zero. This initialization has to be called before any `mytype` variable to be used.

`void dd_clear(x) :`
This is to free the space allocated to a `mytype` variable `x`.

`void dd_set_si(x, a) :`
This is to set a `mytype` variable `x` to the value of signed long integer `a`.

`void dd_set_si2(x, a, b) :`
This is to set a `mytype` variable `x` to the value of the rational expression `a/b`, where `a` is signed long and `b` is unsigned long integers.

`void dd_set_d(x, b) :`
This is to set a `mytype` variable `x` to the value of double `b`. This is available only when the library is compiled without `-DGMPRATIONAL` compiler option.

4.3.2 Arithmetic Operations for `mytype` Numbers

Below `x`, `y`, `z` are of type `mytype`.

`void dd_add(x, y, z) :`
Set `x` to be the sum of `y` and `z`.

`void dd_sub(x, y, z) :`
Set `x` to be the substraction of `z` from `y`.

`void dd_mul(x, y, z) :`
Set `x` to be the multiplication of `y` and `z`.

`void dd_div(x, y, z) :`
Set `x` to be the division of `y` over `z`.

`void dd_inv(x, y) :`
Set `x` to be the reciplocal of `y`.

4.3.3 Predefined Constants

There are several `mytype` constants defined when `dd_set_global_constants(void)` is called. Some constants depend on the double constant `dd_almostzero` which is normally set to 10^{-7} in cdd.h. This value can be modified depending on how numerically delicate your problems are but an extra caution should be taken.

`mytype dd_purezero` :
: This represents the mathematical zero 0.

`mytype dd_zero` :
: This represents the largest positive number which should be considered to be zero. In the GMPRATIONAL mode, it is equal to `dd_purezero`. In the C double mode, it is set to the value of `dd_almostzero`.

`mytype dd_minuszero` :
: The negative of `dd_zero`.

`mytype dd_one` :
: This represents the mathematical one 1.

4.3.4 Sign Evaluation and Comparison for `mytype` Numbers

Below `x`, `y`, `z` are of type `mytype`.

`dd_boolean dd_Positive(x)` :
: Returns `dd_TRUE` if `x` is considered positive, and `dd_FALSE` otherwise. In the GMPRATIONAL mode, the positivity recognition is exact. In the C double mode, this means the value is strictly larger than `dd_zero`.

  `dd_boolean dd_Negative(x)` works similarly.

`dd_boolean dd_Nonpositive(x)` :
: Returns the negation of `dd_Positive(x)`. `dd_Nonnegative(x)` works similarly.

`dd_boolean dd_EqualToZero(x)` :
: Returns `dd_TRUE` if `x` is considered zero, and `dd_FALSE` otherwise. In the GMPRATIONAL mode, the zero recognition is exact. In the C double mode, this means the value is inbetween `dd_minuszero` and `dd_zero` inclusive.

`dd_boolean dd_Larger(x, y)` :
: Returns `dd_TRUE` if `x` is strictly larger than `y`, and `dd_FALSE` otherwise. This is implemented as dd_Positive(z) where `z` is the subtraction of `y` from `x`. `dd_Smaller(x, y)` works similarly.

`dd_boolean dd_Equal(x, y)` :
: Returns `dd_TRUE` if `x` is considered equal to `y`, and `dd_FALSE` otherwise. This is implemented as dd_EqualToZero(z) where `z` is the subtraction of `y` from `x`.

4.3.5 Polyhedra Data Manipulation

`dd_MatrixPtr dd_PolyFile2Matrix (f, err)` :
: Read a Polyhedra data from stream `f` and store it in `matrixdata` and return a pointer to the data.

`dd_MatrixPtr dd_CopyInequalities(poly) :`
Copy the inequality representation pointed by poly to `matrixdata` and return `dd_MatrixPtr`.

`dd_MatrixPtr dd_CopyGenerators(poly) :`
Copy the generator representation pointed by poly to `matrixdata` and return `dd_MatrixPtr`.

`dd_SetFamilyPtr dd_CopyIncidence(poly) :`
Copy the incidence representation of the computed representation pointed by poly to `setfamily` and return `dd_SetFamilyPtr`. The computed representation is `Inequality` if the input is `Generator`, and the vice visa.

`dd_SetFamilyPtr dd_CopyAdjacency(poly) :`
Copy the adjacency representation of the computed representation pointed by poly to `setfamily` and return `dd_SetFamilyPtr`. The computed representation is `Inequality` if the input is `Generator`, and the vice visa.

`dd_SetFamilyPtr dd_CopyInputIncidence(poly) :`
Copy the incidence representation of the input representation pointed by poly to `setfamily` and return `d_SetFamilyPtr`.

`dd_SetFamilyPtr dd_CopyInputAdjacency(poly) :`
Copy the adjacency representation of the input representation pointed by poly to `setfamily` and return `d_SetFamilyPtr`.

`void dd_FreePolyhedra(poly) :`
Free memory allocated to `poly`.

4.3.6 LP Data Manipulation

`dd_LPPtr dd_MakeLPforInteriorFinding(lp) :`
Set up an LP to find an interior point of the feasible region of `lp` and return a pointer to the LP. The new LP has one new variable x_{d+1} and one more constraint: $\max x_{d+1}$ subject to $b - Ax - x_{d+1} \geq 0$ and $x_{d+1} \leq K$, where K is a positive constant.

`dd_LPPtr dd_Matrix2LP(matrix, err) :`
Load `matrix` to `lpdata` and return a pointer to the data.

`dd_LPSolutionPtr dd_CopyLPSolution(lp) :`
Load the solutions of `lp` to `lpsolution` and return a pointer to the data. This replaces the old name `dd_LPSolutionLoad(lp)`.

`void dd_FreeLPData(lp) :`
Free memory allocated as an LP data pointed by `lp`.

`void dd_FreeLPSolution(lps) :`
Free memory allocated as an LP solution data pointed by `lps`.

4.3.7 Matrix Manipulation

`dd_MatrixPtr dd_CopyMatrix(matrix) :`
Make a copy of matrixdata pointed by `matrix` and return a pointer to the copy.

`dd_MatrixPtr dd_AppendMatrix(matrix1, matrix2) :`
: Make a matrixdata by copying `*matrix1` and appending the matrix in `*matrix2` and return a pointer to the data. The colsize must be equal in the two input matrices. It returns a `NULL` pointer if the input matrices are not appropriate. Its `rowsize` is set to the sum of the rowsizes of `matrix1` and `matrix2`. The new matrixdata inherits everything else (i.e. numbertype, representation, etc) from the first matrix.

`int dd_MatrixAppendTo(& matrix1, matrix2) :`
: Same as `dd_AppendMatrix` except that the first matrix is modified to take the result.

`int dd_MatrixRowRemove(& matrix, i) :`
: Remove the ith row of `matrix`.

`dd_MatrixPtr dd_MatrixSubmatrix(matrix, set) :`
: Generate the submatrix of `matrix` by removing the rows indexed by `set` and return a matrixdata pointer.

`dd_SetFamilyPtr dd_Matrix2Adjacency(matrix, err) :`
: Return the adjacency list of the representation given by `matrix`. The computation is done by the built-in LP solver. The representation should be free of redundancy when this function is called. See the function `dd_rowset dd_RedundantRows` and the example program adjacency.c.

4.4 Input/Output Functions

`dd_MatrixPtr dd_PolyFile2Matrix (f, err) :`
: Read a Polyhedra data from stream `f` and store it in `matrixdata` and return a pointer to the data.

`boolean dd_DDFile2File(ifile, ofile, err) :`
: Compute the representation conversion for a polyhedron given by a Polyhedra file ifile, and write the other representation in a Polyhedra file ofile. `*err` returns `dd_NoError` if the computation terminates normally. Otherwise, it returns a value according to the error occured.

`void dd_WriteMatrix(f, matrix) :`
: Write `matrix` to stream `f`.

`void dd_WriteNumber(f, x) :`
: Write `x` to stream `f`. If `x` is of GMP mpq_t rational p/q, the output is p/q. If it is of C double, it is formated as a double float with a decimal point.

`void dd_WritePolyFile(f, poly) :`
: Write tt poly to stream `f` in Polyhedra format.

`void dd_WriteErrorMessages(f, err) :`
: Write error messages given by `err` to stream `f`.

`void dd_WriteSetFamily(f, setfam) :`
: Write the set family pointed by `setfam` to stream `f`. For each set, it outputs its index, its cardinality, a colon ":" and a ordered list of its elements.

`void dd_WriteSetFamilyCompressed(f, setfam) :`
: Write the set family pointed by `setfam` to stream `f`. For each set, it outputs its index, its cardinality or the negative of the cardinality, a colon ":" and the elements in the set or its

complements whichever is smaller. Whenever it outputs the complements, the cardinality is negated so that there is no ambiguity. This will be considered standard for outputing incidence (*.icd, *ecd) and adjacency (*.iad, *.ead) data in cddlib. But there is some minor incompatibility with cdd/cdd+ standalone codes.

`void dd_WriteProgramDescription(f) :`
Write the cddlib version information to stream `f`.

`void dd_WriteDDTimes(f, poly) :`
Write the representation conversion time information on `poly` to stream `f`.

4.5 Obsolete Functions

`boolean dd_DoubleDescription(poly, err) :` (removed in Version 0.90c)
The new function `dd_DDMatrix2Poly(matrix, err)` (see Section 4.2) replaces (and actually combines) both this and `dd_Matrix2Poly(matrix, err)`.

`dd_PolyhedraPtr dd_Matrix2Poly(matrix, err) :` (removed in Version 0.90c)
See above for the reason for removal.

`dd_LPSolutionPtr dd_LPSolutionLoad(lp) :` (renamed in Version 0.90c)
This function is now called `dd_CopyLPSolution(lp)`.

4.6 Set Functions in `setoper` library

The cddlib comes with a simple set operation library `setoper`. The key type defined is `set_type`. A set is represented by a fixed length binary strings. Thus, the maximum length of a set must be declared when it is initialized.

Below the symbols `a`, `b`, `c` are of type `set_type`. The symbols `aP` is a pointer to type `set_type`, and `s`, `t` are of type `long`. Here are some of the functions defined. See `setoper.h` for a complete listing.

`void set_initialize(aP, s) :`
Allocate a `set_type` space of maximum cardinality `s` and make it pointed by `aP`. The set is initialized as empty set.

`void set_free(a) :`
Free the `set_type` space allocated for `a`.

`void set_copy(a, b)) :`
Set `a` to be `b`. The set `a` must be pre-initialized with the same maximum cardinality as that of `b`.

`void set_addelem(a, t)) :`
Add an element `t` to a set `a`. The set `a` stays unchanged if it contains the element `t`.

`long set_card(a)) :`
Return the cardinality of set `a`.

`int set_member(t, a)) :`
Return 1 if `t` is a member of set `a`, and 0 otherwise.

`void set_write(a))` :

Print out the elements of set `a` to `stdout`. The function `void set_fwrite(f, a))` output to stream `f`.

5 An Extension of the CDD Library in GMP mode

Starting from the version 093, the GMP version of cddlib, `libcddgmp.a`, contains all cdd library functions in two arithmetics. All functions with the standard prefix `dd_` are computed with the GMP rational arithmetic as before. The same fuctions with the new prefix `ddf_` are now added to the library `libcddgmp.a` that are based on the C double floating-point arithmetic. Thus these functions are equivalent to `libcdd.a` functions, except that all functions and variable types are with prefix `ddf_` and the variable type `mytype` is replaced by `myfloat`.

In this sense, `libcdd.a` is a proper subset of `libcddgmp.a` and in principle one can do everything with `libcddgmp.a`. See how the new `dd_LPSolve` is written in cddlp.c.

6 Examples

See example codes such as testcdd*.c , testlp*.c, redcheck.c, adjacency.c, allfaces,c and simplecdd.c in the `src` and `src-gmp` subdirectories of the source distribution.

7 Numerical Accuracy

A little caution is in order. Many people have observed numerical problems of cddlib when the floating version of cddlib is used. As we all know, floating-point computation might not give a correct answer, especially when an input data is very sensitive to a small perturbation. When some strange behavior is observed, it is always wise to create a rationalization of the input (for example, one can replace 0.3333333 with 1/3) and to compute it with cddlib compiled with gmp rational to see what a correct behavior should be. Whenever the time is not important, it is safer to use gmp rational arithmetic.

If you need speedy computation with floating-point arithmetic, you might want to "play with" the constant `dd_almostzero` defined in cdd.h:

```
#define dd_almostzero  1.0E-7
```

This number is used to recognize whether a number is zero: a number whose absolute value is smaller than `dd_almostzero` is considered zero, and nonzero otherwise. You can change this to modify the behavior of cddlib. One might consider the default setting is rather large for double precision arithmetic. This is because cddlib is made to deal with highly degenerate data and it works better to treat a relatively large "epsilon" as zero.

Another thing one can do is scaling. If the values in one column of an input is of smaller magnitude than those in another column, scale one so that they become comparable.

8 Other Useful Codes

There are several other useful codes available for vertex enumeration and/or convex hull computation such as lrs, qhull, porta and irisa-polylib. The pointers to these codes are available at

1. lrs by D. Avis [2] (C implementation of the reverse search algorithm [4]).

2. qhull by C.B. Barber [6] (C implementation of the beneath-beyond method, see [10, 20], which is the dual of the dd method).

3. porta by T. Christof and A. Löbel [8] (C implementation of the Fourier-Motzkin elimination).

4. IRISA polyhedral library by D.K. Wilde [23] (C implementation of a variation of the dd algorithm).

5. PPL: the Parma Polyhedra Library [5] by R. Bagnara (C++ implementation of a variation of the dd algorithm).

6. `pd` by A. Marzetta [18] (C implementation of the primal-dual algorithm [7]).

7. Geometry Center Software List by N. Amenta [1].

8. Computational Geometry Pages by J. Erickson [11].

9. Linear Programming FAQ by R. Fourer and J. Gregory [12].

10. ZIB Berlin polyhedral software list:
ftp://elib.zib-berlin.de/pub/mathprog/polyth/index.html.

11. Polyhedral Computation FAQ [13].

9 Codes Using Cddlib

There are quite a few nice programs using some functions of cddlib. Here are some of them.

1. `LattE` [9] computes the number of lattice points in a convex polytope.

2. `Minksum` [22] is a program to compute the V-representation (i.e. the set of vertices) of the Minkowski addition of several convex polytopes given by their V-representation in $\mathbb{R}^d$. It is an implementation in C++ language of the reverse search algorithm [14] whose time complexity is polynomially bounded by the sizes of input and output.

3. `Gfan` [17] is a program to list all reduced Gröbner bases of a general polynomial ideal given by a set of generating polynomials in n-variables. It is an implementation in C++ language of the reverse search algorithm [15].

4. `TOPCOM` [21] computes the combinatorial structure (the oriented matroid) of a point configuration and enumerates all triangulations of a point set. It detects the regularlity of a triangulation using cddlib.

Acknowledgements.

I am grateful to Tom Liebling who provided me with an ideal opportunity to visit EPFL for the academic year 1993-1994. Later, Hans-Jakob Lüthi (ETHZ) and Emo Welzl (ETHZ) joined to support the the development of cdd codes (cdd, cdd+, cddlib). Without their generous and continuing support, the present form of this program would not have existed.

There are many other people who helped me to improve cdd, in particular, I am indebted to David Avis, Alexander Bockmayr, David Bremner, Henry Crapo, Istvan Csabai, Francois Margot, Marc Pfetsch, Alain Prodon, Jörg Rambau, Dima Pasechnik, Shawn Rusaw, Matthew Saltzman, Masanori Sato, Anders Jensen, Ruriko Yoshida, Charles Geyer (listed in arbitrary order) and those listed in the HISTORY file.

References

[1] N. Amenta. Directory of computational geometry. http://www.geom.uiuc.edu/software/cglist/.

[2] D. Avis. *lrs Homepage.* http://cgm.cs.mcgill.ca/~avis/C/lrs.html.

[3] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms. *Computational Geometry: Theory and Applications*, 7:265–302, 1997.

[4] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.*, 8:295–313, 1992.

[5] R. Bagnara. Parma polyhedra library homepage, 2004. http://www.cs.unipr.it/ppl/.

[6] C.B. Barber, D.P. Dobkin, and H. Huhdanpaa. *qhull, Version 2003.1*, 2003. program and report available from http://www.qhull.org/.

[7] D. Bremner, K. Fukuda, and A. Marzetta. Primal-dual methods for vertex and facet enumeration. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 49–56, 1997.

[8] T. Christof and A. Löbel. PORTA: Polyhedron representation transformation algorithm (ver. 1.3.1), 1997. http://www.zib.de/Optimization/Software/Porta/.

[9] J. de Loera, D. Haws, R. Hemmecke, Peter Huggins, J. Tauzer, and R. Yoshida. *LattE.* University of California, Davis, 2005. available from http://www.math.ucdavis.edu/ latte/.

[10] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, 1987.

[11] J. Erickson. Computational geometry pages, list of software libraries and codes. http://compgeom.cs.uiuc.edu/~jeffe/compgeom/.

[12] R. Fourer and J.W. Gregory. Linear programming frequently asked questions (LP-FAQ). http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html.

[13] K. Fukuda. Polyhedral computation FAQ, 1998. Both html and ps versions available from http://www.ifor.math.ethz.ch/~fukuda/fukuda.html.

[14] K. Fukuda. From the zonotope construction to the Minkowski addition of convex polytopes. *Journal of Symbolic Computation*, 38(4):1261–1272, 2004. pdf file available from http://www.cs.mcgill.ca/~fukuda/download/paper/minksum031007jsc.pdf.

[15] K. Fukuda, A. Jensen, and R. Thomas. Computing Gröbner fans. Technical report. In preparation.

[16] K. Fukuda and A. Prodon. Double description method revisited. In M. Deza, R. Euler, and I. Manoussakis, editors, *Combinatorics and Computer Science*, volume 1120 of *Lecture Notes in Computer Science*, pages 91–111. Springer-Verlag, 1996. ps file available from ftp://ftp.ifor.math.ethz.ch/pub/fukuda/reports/ddrev960315.ps.gz.

[17] A.N. Jensen. *Gfan version 0.1: A User's Manual.* Department of Mathematical Sciences, University of Aarhus and Institute for Operations Research, ETH Zurich, 2005. available from http://home.imf.au.dk/ajensen/software/gfan/gfan.html.

[18] A. Marzetta. *pd – C-implementation of the primal-dual algoirithm*, 1997. code available from http://www.cs.unb.ca/profs/bremner/pd/.

[19] T.S. Motzkin, H. Raiffa, GL. Thompson, and R.M. Thrall. The double description method. In H.W. Kuhn and A.W.Tucker, editors, *Contributions to theory of games, Vol. 2.* Princeton University Press, Princeton, RI, 1953.

[20] K. Mulmuley. *Computational Geometry, An Introduction Through Randamized Algorithms.* Prentice-Hall, 1994.

[21] J. Rambau. *TOPCOM, a package for computing Triangulations Of Point Configurations and Oriented Matroids.* University of Bayreuth, 2005. available from http://www.uni-bayreuth.de/departments/wirtschaftsmathematik/rambau/TOPCOM/.

[22] C. Weibel. *Minksum version 1.1.* Mathematics Institute, EPF Lausanne, 2005. available from http://roso.epfl.ch/cw/poly/public.php.

[23] D.K. Wilde. A library for doing polyhedral operations. Master's thesis, Oregon State University, Corvallis, Oregon, Dec 1993. Also published in IRISA technical report PI 785, Rennes, France; Dec, 1993.