Computer Science
University of Manchester

MathPIP: A Mathematica Interface for PIP
User's Guide and Reference Manual

Zbigniew Chamski

Technical Report 94-6-1

MathPIP: A Mathematica™ Interface for PIP User's Guide and Reference Manual *

Zbigniew Chamski

Department of Computer Science
University of Manchester
Oxford Road, Manchester, UK.
chamskiz@cs.man.ac.uk

Preliminary version (distributed with MathPIP)

20 June 1994

Abstract

The Parametric Integer Programming (PIP) algorithm due to Feautrier is highly suitable for solving linear programming problems arising in program parallelization and synthesis. In this report, we describe the interface allowing to use the arbitrary-precision arithmetic implementation of PIP as an external problem solver in Mathematica, thus providing a convenient means for fast experimentation with algorithms which use PIP as a building block. The first part of the report offers a short overview of PIP, followed by an example-based user's guide to MathPIP. The second part is a reference manual describing in detail each user-visible component of MathPIP.

Recent technical reports issued by the Department of Computer Science, Manchester University, are available by anonymous ftp from ftp.cs.man.ac.uk in the directory pub/TR. The files are stored as PostScript, in compressed form, with the report number as filename. They can also be obtained on WWW via URL http://www.cs.man.ac.uk/csonly/cstechrep/index.html. Alternatively, all reports are available by post from The Computer Library, Department of Computer Science, The University, Oxford Road, Manchester M13 9PL, UK.

1 Introduction

The MATHEMATICA software system ([15]) appears as a convenient framework for the implementation of novel algorithms and the experimentation with program rewriting. The combined power of a single programming language and a convenient interface to external user-written applications allows performance limitations, inherent to the interpreter-based operation of the system, to be overcome. Interfacing facilities offered by MATHLINK ([13]) are particularly useful when implementing and evaluating techniques based on computation-intensive building blocks, such as linear algebra algorithms. The usefulness of this approach has been successfully shown in a MATHEMATICA-based reimplementation of the ALPHA DU CENTAUR environment ([12]) at Irisa (Rennes, France), which extensively uses the Chernikova's algorithm ([4, 11]) to generate irredundant representations of polyhedra.

The PIP algorithm, due to Feautrier ([6, 7, 8, 5, 2]), is another linear programming algorithm frequently used in the context of parallel program transformation and code generation. We ported its original implementation ([8]) to arbitrary-precision arithmetic ([2]) using the publicly available GNU MP library ([10]) and demonstrated its practical suitability and robustness in the context of the enumeration of iteration domains. The versatility of the algorithm and the performance of the MULTIPIP implementation incited us to design and implement an interface for the interactive use of PIP from inside MATHEMATICA, aiming at the fast prototyping of PIP-based algorithms.

The design choices were motivated by the properties and the behaviour of the algorithm. It has been demonstrated ([1, 2]) that it can be used in parallelizing the process of loop generation. Also, in less structured applications (e.g., exact dependence graph construction), several independent PIP problems may be ready for treatment almost simultaneously. Therefore, the possibility of having several instances of a PIP request server running in parallel was the key feature that had to be provided.

The user interface is based on our previous experience with interactive use of user-written functionalities in MATHEMATICA. The pattern-matching features are used to limit the burden of unnecessary (and sometimes harmful) typing by introducing the concept of a default PIP process. Input problems and solutions are represented using specific non-rewritable expressions, which can be viewed as abstract syntax trees. This ensures that input data and results will remain stable under the default MATHEMATICA evaluation rules and hence, that an accidental evaluation of these objects will not alter them.

In the rest of this report, we first give an overview of PIP, then we describe the user-visible functionalities of MATHPIP and their use, and provide the reference manual of the interface. We conclude with the directions for contributing bug fixes and improvements.

2 PIP: an Overview

PIP stands for *Parametric Integer Programming*, and is aimed at the determination of an integer minimum of a convex polyhedron defined using a system of affine constraints which may linearly depend on a set of integer parameters.

The cost function is the ascending lexicographic ordering; it is not linear, but proves extremely useful when solving practical problems arising in program transformation and synthesis, and suppresses the risk of degeneration during the solving process ([6, 14]).

In more formal terms, a PIP problem consists in finding the lexicographic minimum of a parametric polyhedron

$$\mathcal{F}(z) = \{x \mid Ax + Bz + c \geq 0, Kz + h \geq 0, x \geq 0, z \geq 0\} \tag{1}$$

where x denotes a *vector* of variables and z is a *vector* of parameters. The system $Ax + Bz + c \geq 0$ forms the *constraints* of the problem, and the system $Kz + h \geq 0$ is called the problem's *context*. (NB: the notation $x \geq 0$ where $x = (x_1 \ldots x_n)$ is a vector is understood as $x_i \geq 0\ \forall i,\ 1 \leq i \leq n$.) A, B and K are integer matrices, and c and z are vectors of integers.

Since the problem can obviously depend on the values of the parameters, the solutions reflect this through conditional expressions separating the cases where two different solutions are found depending on whether an affine constraint on the parameters holds or not. These conditionals can be nested, leading to solutions in the form of *selection trees*.

The problem of integer solutions is handled through the dynamic introduction of new parameters, defined as the result of an integer division of an affine combination of existing ones by a positive integer. The visibility of a new parameter is restricted to the subtree for which it was introduced.

Since the only problem that can be solved is the search of the lexicographic *minimum*, it is necessary to convert a maximization problem to a suitable minimization one. This is done by introducing a slack "big parameter" *big* and replacing each variable x_i by $big - x_i$. Hence, in the minimization problem corresponding to the original one, the coefficient of the big parameter in each constraint (see above) is the sum of the original variable coefficients in that constraint, while all variable coefficients are negated. The original context is extended to accommodate an additional parameter while leaving its coefficients null (there is no constraints on the value of the big parameter, except for the implicit non-negativity constraint.)

If the result of a maximization problem does not depend, for some component of x, on the value of the big parameter (and therefore, is uniformly null,) then the corresponding component is unbounded.

Applications of PIP include problems such as exact dependence testing and polyhedra scanning problems (taken in a broad sense: determination of accessed areas in arrays, generation of loop bounds etc.) The interested reader can consult [6, 7, 8, 9, 1, 5, 2, 3] for further information on the algorithm and its applications.

The detailed description of the algorithm and the proofs of its properties can be found in [6] and [5]. References [7] and [8] (in French) provide information on translating real-life problems into PIP input. Loop generation techniques based on PIP are discussed in [7, 1, 5, 2, 3]. In addition, [2] contains a performance comparison between MultiPIP and an implementation of the Fourier-Motzkin pairwise elimination in the context of bound generation.

3 User's guide

NB: in the following, we assume that the reader is familiar with MATHEMATICA and PIP.

The core of the MATHPIP server is a modified version of MULTIPIP ([2]), an arbitrary-precision arithmetic version of the original PIP software ([8]) written by Feautrier and Tawbi. The PIP process communicates with MATHEMATICA using the MATHLINK protocol and facilities.

MATHPIP is supplied as a set of C source files, together with a directly loadable MATHEMATICA package. Please refer to section 4 for the detailed installation procedure. For the rest of this section, we will assume that the binary part has been successfully built and installed.

All symbols used by the interface appear in the context `"Pip‘"`. This context is automatically added to the current `$ContextPath` when the package `"Pip.m"` is loaded.

3.1 Starting and Stopping a MATHPIP Server

To start a default MATHPIP server, start MATHEMATICA and simply load the package `"Pip.m"` using '`Get`' or '`<<`'. This will start the default server '`defaultPip`' on the default host, i.e., the machine on which your MATHEMATICA session runs.

To stop the default server, simply type

```
stop[]
```

You can start additional servers by typing

serverId `= start[`*host*`]`

where *serverId* is a symbol that will be used to identify the server in further operations, and *host* is the name of the machine on which you want the server to run, surrounded by double quotes. Omitting this name will start the server on the default host. If you want to run MATHPIP on remote hosts in a heterogeneous environment, you must first ensure that there is an appropriate binary for each type of hardware platfor you will use.

To stop a named server, type

`stop[`*serverId*`]`

where *serverId* is the name of a previously started server. If the name given does not correspond to an existing server, or if the server connection has closed down, MATHEMATICA will issue a warning message.

3.2 Input Data Structures

The components of a MathPIP problem are wrapped into a composite expression with head 'problem'. The general pattern of a MathPIP problem in Mathematica is

```
problem[numOfVars_Integer,
        numOfParms_Integer,
        bigParmPos_Integer,
        integerSolution:(True|False),
        constraints:{{_Integer..}..},
        context:({}|{{_Integer..}..})]
```

where:

- numOfVars is the number of variables in the problem;
- numOfParms is the number of parameters of the problem;
- bigParmPos is the position of the "big parameter" counted from 1 (as opposed to zero in "standard" PIP);
- integerSolution is the symbol True if integer solutions are requested, and False otherwise;
- constraints is an integer matrix representing the constraints of the problem;
- context is an empty list, or an integer matrix representing the context of the problem.

The layout of the constraint and context matrices is the same as in standard PIP and MultiPIP, namely, a constraint contains numOfVars variable coefficients, followed by the constant (affine) coefficient and the numOfParms parameter coefficients. A context line contains numOfParms parameter coefficients followed by the constant (affine) term.

3.3 Output Data Structures

The expressions returned by the MathPIP server are very close to those produced by standard PIP. The differences are in the names of operators and in the syntax of the "newparm" operator.

The revised output grammar is as follows:

<result>	::=	voidContext | *<quast>*
<quast>	::=	if[*<vector>*, *<quast>*, *<quast>*]
		| newParm[_Integer, *<divterm>*, *<quast>*]
		| {*<vector>*..}
		| empty
<divterm>	::=	div[*<vector>*, _Integer]
<vector>	::=	{*<number>* ..}
<number>	::=	(_Integer|_Rational)

The symbol 'voidContext' indicates that the *context* of the problem is not feasible. A non-feasible *solution* is indicated by the symbol 'empty' appearing in a leaf of the selection tree.

Unlike in the original output grammar of PIP, all output constructs have a fixed arity. A list of <*quast*>s, which was legal in the original grammar, is *not* in MATHPIP, and is replaced by a nest of 'newParm' operators, *always* ternary. All subsequent 'newParm's appear in the third operand of the enclosing one, while the original grammar allowed a flat sequence of "newparm" declarations to precede a *quast*.

The terms of the form 'div[*vect*, *divisor*]' express an integer division of the expression *vect*.$(z_1, \ldots, z_{last}, 1)$ by *divisor*. The parameter vector includes all "new parameters" that are visible at the current node of the selection tree.

Conditional solutions of the form 'if[*cond*, *trueterm*, *falseterm*]' evaluate to *trueterm* if the constraint *cond*.$(z_1, \ldots, z_{last}, 1) \geq 0$ holds, and to *falseterm* otherwise. The constraint can involve any dynamically introduced "new parameter" whose scope includes the conditional.

Feasible affine solutions are represented by a matrix of mixed Integer and Rational values.

3.4 Solving a Problem

To solve the parametric problem

$$\text{lexmin}(\{x_1 \mid x_1 + p_1 - 4 \geq 0,\ x_1 - p_1 + 2 \geq 0,\ x_1 + p_1 - 8 \leq 0,\ x_1 - p_1 - 2 \leq 0\}) \quad (2)$$

(fig. 1) in the context $\{p_1 \geq 1,\ p_1 \leq 5\}$ using the default server 'defaultPip', type

```
solveMin[problem[1,1,-1,True,
                {{1,-4,1},{1,2,-1},
                 {-1,8,-1},{-1,2,1}},
                {{1,-1},{-1,5}}]]
```

at MATHEMATICA prompt. The operands of 'problem' indicate that the input has one variable, one parameter, no big parameter (so it is a minimization problem,) that it has to be solved in integers, and supply the constraint system and the context in which the problem will be solved. The result of the evaluation will be the expression

```
if[{1, -4}, {{1, -2}}, {{-1, 4}}]
```

In human-readable form, it indicates that for $p_1 \geq 4$ the minimum of x_1 is $p_1 - 2$, and $-p_1 + 4$ otherwise. Since the parameters are supposed to be integers, the "else" case can be speficied by a weak inequality and corresponds to $p_1 \leq 3$ (see fig. 2.)

The maximization problems can be solved directly by calling the function 'solveMax' in the same way. 'solveMax' accepts both minimization and maximization problems; the minimization ones are converted to the maximization form (see section 3.7) before being sent to the server.

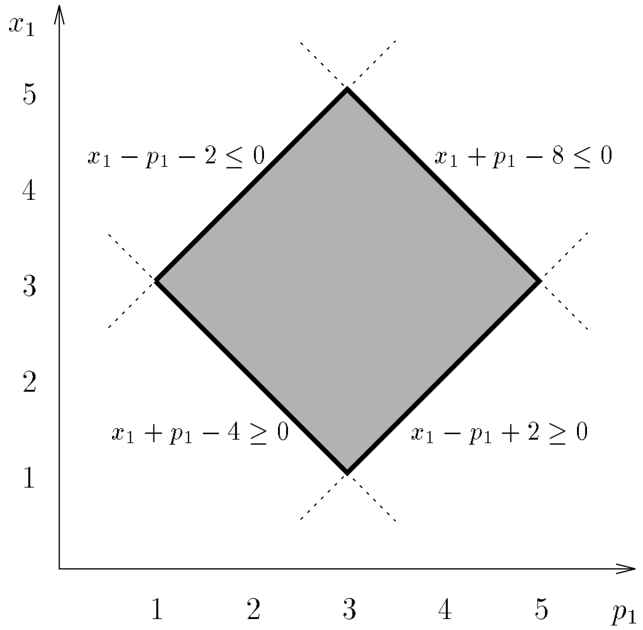

Figure 1: A sample domain (see equation 2)

Adding a MathPIP server name as the first parameter of either '`solveMin`' or '`solveMax`' will cause the input problem to be solved by the specified server.

Both functions perform an extensive dynamic checking of the input data and server specification. This helps in fixing problems with the structure of input data or the tools that generate it, but can lead to a noticeable run-time overhead, particularly in the case of small problems. If you are sure about the consistency of your input, you can directly use the main interface function '`solve`' with the same operands.

'`solve`' assumes that your input problem is consistent, generates any additional information required to perform the computation (default server name, omitted "big parameter" position) and passes the resulting data to the appropriate PIP server.

3.5 Asynchronous Operation

The asynchronous communication primitives '`send`' and '`receive`' allow you to send input data to different PIP servers, or send different sets of input data successively to the same server, and receive the results when they are needed and available. Function '`break`' interrupts a long computation and clears all internal data structures in the server so that it is ready for a new computation.

'`send`' and '`receive`' only accept fully specified problems and perform no parameter checking, neither on the name of the specified server nor on the data being sent. '`send`' returns immediately after sending the data, and returns no result if the operation was successful. If not, any values generated by the system are passed to the user or the

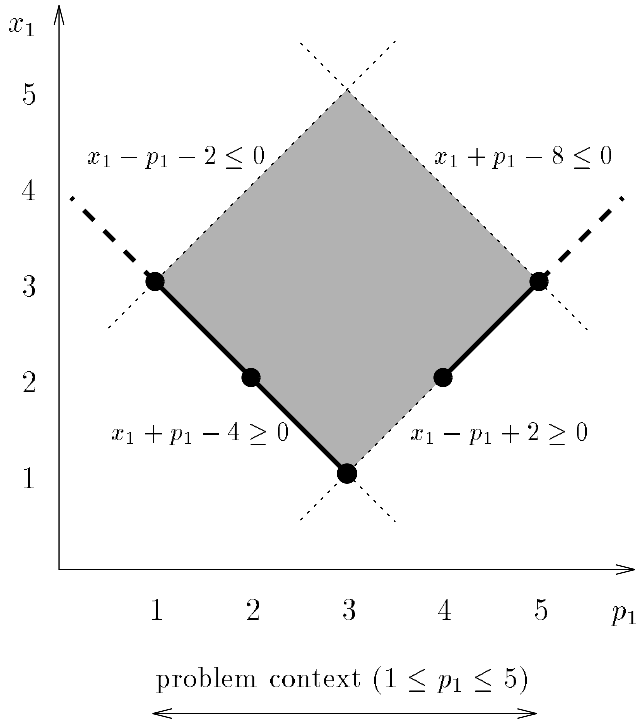

Figure 2: Interpretation of the solution of problem (2)

function which called '`send`', and a warning message is printed on the screen.

As for '`solveMin`', '`solveMax`' and '`solve`', the server identification can be omitted, in which case server '`defaultPip`' will be assumed.

The result returned by '`receive`' depends on whether or not there was data available on the link. If the PIP server generated a consistent (balanced) expression, it will be returned directly. If the data were truncated (due to a '`break`' or link shutdown), '`receive`' will return the symbol '`System‘$Failed’`. If a '`break`' operation was performed and '`System‘$Failed`', the following call to '`receive`' will return the symbol '`System‘$Aborted`'. Finally, if there is no data whatsoever available on the link, '`receive`' returns immediately the symbol '`noData`'.

To solve two problems '`pb1`' and '`pb2`' in parallel on hosts '`rdf040`' and '`rdf041`', you can use the following sequence of MATHEMATICA instructions:

```
server1 = start["rdf040"]
server2 = start["rdf041"]
send[server1, pb1]
send[server2, pb2]

(* perform any computations  that  DO  NOT *)
(* depend on the results; when you finally *)
```

```
(* need them, run the following loops:     *)

result1 = Block[{res1 = receive[server1]},
                While[res1 == noData,
                      res1 = receive[server1]];
                res1]

result2 = Block[{res2 = receive[server2]},
                While[res2 == noData,
                      res2 = receive[server2]];
                res2]
```

3.6 Profiling Information

Functions 'problemTimeStats' and 'totalTimeStats' allow the user to extract the information on the time spent solving the last problem or the total CPU time spent (in both user and system routines) by the specified PIP server, together with the number of cross-products (elementary pivoting steps) performed and the number of problems treated to date. Aborted computations are accounted for.

Example:

```
In[28]:= solveMin[problem[1,1,-1,True,
                 {{1,-4,1},{1,2,-1},
                  {-1,8,-1},{-1,2,1}},
                 {{1,-1},{-1,5}}]]

Out[28]= if[{1, -4}, {{1, -2}}, {{-1, 4}}]

In[29]:= problemTimeStats[]

Out[29]= {0.01, 0.00999999999999998, 73}

In[30]:= totalTimeStats[]

Out[30]= {8, 0.2, 0.19, 341}
```

The result produced by 'problemTimeStats' indicates that solving the last problem on the default PIP server took 10 ms of user time and nearly 10 ms of system time, and required 73 cross products. The global information indicates that eight problems have been solved so far by the default server, that the total user time is 200 ms, the total system time is 190 ms, and that a total of 341 cross products was performed.

Two other functions: 'problemValueStats' and 'totalValueStats' supply the information on the largest values manipulated during the computations. Since the intermediate coefficients can be far beyond the limits of the 32-bit "machine integer" representation, they are supplied in string form. This makes the display of the values easier, while still allowing the conversion to arbitrary-precision integers of MATHEMATICA. The information returned by both functions consists of three values: the largest numerator term in a cross-product computation, the largest common denominator of a tableau, and the largest numerator obtained after a cross-product.

'`problemValueStats`' returns the values for the last problem processed by the specified server, while '`totalValueStats`' returns the largest values manipulated since the startup of the server.

Example:

```
In[14]:= send[pb4min]

In[15]:= receive[]

Out[15]= if[{-1, 0, 0, 0, 3}, if[{0, -1, 0, 0, 3},
   if[{0, 0, -1, 0, 3}, if[{0, 0, 0, -1, 3},
     if[{-400, 0, 0, 0, 1}, if[{0, -500, 0, 0, 1},
       if[{0, 0, -250, 0, 1}, if[{0, 0, 0, -2000, 3},

   .... etc ....

In[16]:= problemValueStats[]

Out[16]= {539310000180000000000000000000000,
600000000000000000, 629550000180000000000}

In[17]:= MapThread[Print,
                  {{"Max numerator term:\t\t",
                    "Max denominator term:\t\t",
                    "Max cross-product numerator:\t"},
                  Out[16]}];
Max numerator term:             539310000180000000000000000000000
Max denominator term:           600000000000000000
Max cross-product numerator:    629550000180000000000
```

3.7 Problem Conversions

Problem conversions proposed in this version cover the automatic generation of maximization problems, and the "rotation" of problems used in loop generation to move to the next (or previous) loop index. Users are invited to submit their own conversion functions if they find them appropriate for a large distribution.

3.7.1 Min-to-max Conversions

When using the low-level functions '`solve`', '`send`' and '`receive`', you will need to explicitly convert the maximization problems to minimization ones, since the PIP kernel can only solve the latter form. Function '`minToMaxPb`' implements the conversion mechanism proposed in [8], returning a minimization problem. E.g., the conversion of the problem

```
problem[1, 1, -1, True,
        {{1,-4,1},{1,2,-1}, {-1,8,-1},{-1,2,1}},
        {{1,-1},{-1,5}}]
```

to the maximization form returns

```
problem[1, 2, 4, True,
        {{-1,-4,1,1},{-1,2,-1,1},
         {1,8,-1,-1},{1,2,1,-1}},
        {{1,0,-1},{-1,0,5}}]
```

corresponding to

$$\underset{1 \le p_1 \le 5}{\text{lexmin}} \left(\{(big - x_1) \;\; | \;\; (big - x_1) + p_1 - 4 \ge 0,\ (big - x_1) - p_1 + 2 \ge 0,\right.$$
$$\left.(big - x_1) + p_1 - 8 \le 0,\ (big - x_1) - p_1 - 2 \le 0\}\right),$$

where *big* stands for the unconstrained "big parameter" of [8]. Solving the resulting minimization problem using '`solve`' gives

```
if[{1,0,-3},{{1,1,-8}},{{-1,1,-2}}]
```

corresponding to

$$x_1 \le (\mathit{if}\; p_1 \ge 3 \;\mathit{then}\; -p_1 + 8 \;\mathit{else}\; p_1 + 2).$$

Function '`maxToMinPb`' performs the dual conversion: from the input implementing a maximization problem to that implementing a minimization over the same polyhedron.

3.7.2 Problem Rotation

Function '`nextMinPb`' generates a "rotated" problem in which the leftmost variable is removed from its original location and appended at the end of the parameter vector. Similarly, function '`prevMinPb`' removes the last parameter from the parameter vector and prepends it as the first component of the variable vector.

Consider the set of constraints defining the polyhedron shown in fig. 1. Using functions '`nextMinPb`' and '`minPbToMaxPb`' it is possible to derive from the problem '`pb0min`' defining the (constant) minimum of (p_1, x_1) the problems defining the maximum of (p_1, x_1) (`pb0max`) and the extrema of x_1 (resp. `pb1min` and `pb1max`) as functions of p_1:

```
In[38]:= pb0min = problem[2,0,-1,True,
                          {{1,1,-4}, {-1,1,2},
                           {-1,-1,8}, {1,-1,2}}, {}]

Out[38]= problem[2,0,-1,True,
                 {{1,1,-4}, {-1,1,2},
                  {-1,-1,8}, {1,-1,2}}, {}]

In[39]:= pb0max = minToMaxPb[pb0min]

Out[39]= problem[2,1,4,True,
                 {{-1,1,-4,2}, {1,-1,2,0},
                  {1,1,8,-2}, {-1,1,2,0}}, {}]

In[40]:= pb1min = nextMinPb[pb0min]

Out[40]= problem[1,1,-1,True,
                 {{1,-4,1}, {1,2,-1},
                  {-1,8,-1}, {-1,2,1}}, {}]

In[41]:= pb1max = minToMaxPb[pb1min]

Out[41]= problem[1,2,4,True,
                 {{-1,-4,1,1}, {-1,2,-1,1},
                  {1,8,-1,-1}, {1,2,1,-1}}, {}]
```

This set of primitives allows to build loop generators in a straightforward way. However, the manipulation of the context when rotating a problem is left to the user in order to keep the basic functions as simple as possible.

4 Reference Guide

In this section, we provide a more structured information, to be used as reference for installing and using MathPIP. Please feel free to send all information on omitted, unclear or erroneous points in this section to the author (see section 5.)

4.1 Requirements

First of all, you need the Mathematica software, version 2.1 or above. Your distribution must include the MathLink interface generator, and in particular, the MathLink runtime library, to be found in

<math-directory>`/Bin/MathLink/libML.a`

If you never installed MultiPrecision PIP before, you will also need to get, build and install the GNU MP library. It can be found on most FTP sites carrying GNU software. The archive file containing the current version is '`gmp-1.3.2.tar.gz`'.

4.2 Installation

To install the interface code, first build the executable external program '`mathpip`' on your system using '`make`' and the `Makefile` supplied with the sources. You will need to adjust the variables defining the locations of the directories containing the GNU MP library, the MATHEMATICA include files and the MATHLINK library.

In the `Makefile`, check and modify if necessary the values of variables `INCDIR`, `LIBDIR` and `MATHDIR`. Their respective functions are as follows:

- `INCDIR` designates the directory containing the GNU MP header files;
- `LIBDIR` indicates the location of the library '`libgmp.a`' or its specific versions (with names derived from '`libgmp$(EXT).a`') used for profiling, version management etc.;
- `MATHDIR` contains the name of the root directory of the MATHEMATICA distribution.

Once the binary file '`mathpip`' is built, make it visible to the shell so that when the default shell is invoked, '`mathpip`' will be found in the default command path. You can achieve this either by modifying the definition of the `path` variable in your initialization file, or by making '`mathpip`' appear in your default directory of executable files. Make sure that the changes have come into effect before you proceed to the next step.

Start MATHEMATICA, and load (using '`Get`' or '`<<`') the package `"Pip.m"`. This will start a default PIP server and bind its identification (a MATHEMATICA `LinkObject`) to the symbol '`defaultPip`'. If the binary can not be found, MATHEMATICA will issue an error message of the form

```
sh: mathpip: Permission denied
LinkConnect::linkc: LinkObject[mathpip, 4, 1] is dead;
    attempt to connect failed.
```

Test the operation of the interface on the examples from previous section. When correct, add `"Pip.m"` to your package hierarchy and add the line

```
Needs["Pip`"]
```

to your `$HOME/init.m` file. Alternatively, you can use the full form of `Needs`:

```
Needs["Pip`", "Pip-package-file"]
```

where *"Pip-package-file"* is the full name of the file containing the package, or even load it explicitly using '`Get`'.

Other names can be used for the binary file, provided the value of variable '`binaryName`' (see page 13) is properly set inside your MATHEMATICA session before loading `"Pip.m"`.

4.3 Concepts

PIP problem

A *PIP problem* is a the formal specification of the polyhedron whose lexicographical minimum is to be determined using PIP.

■ For the purpose of MATHPIP, problems are coded as MATHEMATICA expressions with head '`problem`'. ■ See page 4.

server

A *server* is a PIP process communicating with a MATHEMATICA session over MATHLINK.

■ A server can solve PIP *problems* coded as MATHEMATICA expressions. ■ All servers are independent from each other. ■ A server is identified by the `LinkObject` associated with the connection between this server and the MATHEMATICA session.

default server

The *default server* is the server used when no explicit server name is given to functions requiring a running PIP server.

■ A *default server* should be maintained alive at all times. ■ See also: variable '`defaultPIP`'.

default host

The *default host* is the machine on which the MATHEMATICA session runs.

4.4 Contexts

`"Pip`"`

`"Pip`"` is the context in which appear all standard symbols related to MATHPIP.

■ This context is automatically defined when the package "`Pip.m`" is loaded.

4.5 Variables

`defaultPip`

`defaultPip` designates the default PIP server to use when an explicit name is omitted.

■ After the package "`Pip.m`" has been loaded, '`defaultPip`' evaluates to the `LinkObject` associated with the default PIP server running on the default host. ■ If the original default server dies, you must use '`defaultPip = start["`*host*`"]`' to define a new default server. ■ See also: function '`start`'.

`binaryName`

`binaryName` is a `String` variable evaluating to the name of the executable file containing the code of MathPIP.

■ The default value of '`binaryName`' is `"mathpip"`. ■ The file must be found in the current command path and must have the '`execute`' permission set. ■ If '`binaryName`' is set to a `String` value before the package `"Pip`"` is loaded, its value overrides the default.

4.6 Functions

The set of available functions is subdivided into categories according to their purpose. All but the conversion functions require a running MathPIP server.

4.6.1 Starting and Stopping MathPIP Servers

`start["`*host*`"]`

`start["`*host*`"]` initializes ("`Install`"-s) a new MathPIP server on the specified *host*.

■ If the host name is omitted, the default host is assumed. ■ See page 3. ■ See also: variables '`defaultPIP`' and '`binaryName`', function '`stop`'.

`stop[`*serverId*`]`

`stop[`*serverId*`]` stops ("`Uninstall`"-s) the specified server.

■ If the server identification *serverId* is omitted, '`defaultPip`' is assumed. ■ See page 3.
■ See also: variable '`defaultPIP`', function '`start`'.

4.6.2 Synchronous Problem Solving

`solve[`*serverId*`, `*pb*`]`

`solve[`*serverId*`, `*pb*`]` solves problem *pb* on server *serverId* without checking for input data coherence nor server appropriateness.

■ If the server identification is omitted, '`defaultPip`' is assumed. ■ '`solve`' returns the control when a result packet has been received from *serverId*. ■ The result is '`System`$Failed`' or '`System`$Aborted`' if the computation was interrupted by an abort operation ('`C-c a`').
■ See page 6. ■ See also: variable '`defaultPip`'.

`solveMin[`*serverId*`, `*pb*`]`

`solveMin[`*serverId*`, `*pb*`]` solves problem *pb* on server *serverId* checking for input data coherence and server appropriateness.

■ If the server identification is omitted, 'defaultPip' is assumed. ■ 'solveMin' returns the control when a result packet has been received from *serverId*. ■ The result is the symbol 'System`$Failed' or 'System`$Aborted' if the computation was interrupted by an abort operation ('C-c a'). If a call to 'solve' is aborted, all data received from the server up to the first occurrence of 'System`$Aborted' are potentially invalid and should be discarded. ■ If the LinkObject supplied as an explicit server name is not connected to the application specified by 'binaryName', 'solveMin' returns System`$Failed. ■ Similarly, it returns System`$Failed if the input problem fails to match several coherency constraints, e.g., when the constraint set or a non-empty context do not form an integer matrix, or when the numbers of variables and parameters are incompatible with the matrices. ■ See page 5. ■ See also: variables 'defaultPip' and 'binaryName'.

solveMax[*serverId*, *pb*]

solveMax[*serverId*, *pb*] solves the maximization form of problem *pb* on server *serverId* checking for input data coherence and server appropriateness.

■ In addition to the functionalities of 'solveMin', 'solveMax' converts the input problem to its maximization form if it does not involve a "big parameter". ■ See page 5. ■ See also: variables 'defaultPip' and 'binaryName', function 'solveMin'.

4.6.3 Asynchronous Problem Solving

send[*serverId*, *pb*]

send[*serverId*, *pb*] asynchronously sends problem *pb* to server *serverId*.

■ If the server identification is omitted, 'defaultPip' is assumed. ■ 'send' performs no checks on the parameters. ■ If the operation is successful, 'send' returns no result. If the server cannot be reached, 'send' returns 'System`$Failed'. ■ See page 6. ■ See also: variable 'defaultPip'.

receive[*serverId*]

receive[*serverId*] checks the availability of results from *serverId* and returns the data if available.

■ If the server identification is omitted, 'defaultPip' is assumed. ■ If the results are not available, 'receive' returns the symbol noData. ■ If an abort operation was performed (using either 'C-c a' or 'break',) all packets received from *serverId* up to the first occurrence of the symbol 'System`$Aborted' are potentially invalid and should be discarded. ■ See page 6. ■ See also: variable 'defaultPip'.

break[*serverId*]

break[*serverId*] interrupts the computation being performed by server *serverId*.

■ If the server identification is omitted, 'defaultPip' is assumed. ■ The end of invalid result data is marked by the symbol 'System`$Aborted'. ■ See page 6. ■ See also: variable 'defaultPip'.

4.6.4 Profiling Information

`problemTimeStats[`*serverId*`]`

`problemTimeStats[`*serverId*`]` returns the information on the computational resources used to solve the last problem on server *serverId*.

■ If the server identification is omitted, 'defaultPip' is assumed. ■ 'problemTimeStats' returns a list containing the amount of user and system time spent and the number of cross products performed while solving the last problem, in that order. ■ The result is meaningful even if the computation was aborted. ■ See page 8.

`totalTimeStats[`*serverId*`]`

`totalTimeStats[`*serverId*`]` returns the information on the computational resources used to solve all problems solved so far on server *serverId*.

■ If the server identification is omitted, 'defaultPip' is assumed. ■ 'totalTimeStats' returns a list containing the total number of problems solved, the amount of user and system time spent and the total number of cross products performed while solving all problems treated so far by the specified server, in that order. ■ The result takes into account all computations, including aborted ones. ■ See page 8.

`problemValueStats[`*serverId*`]`

`problemValueStats[`*serverId*`]` returns the information on the largest coefficient and subterm values reached while solving the last problem on server *serverId*.

■ If the server identification is omitted, 'defaultPip' is assumed. ■ 'problemValueStats' returns a list of `String` objects representing the textual form of the absolute values of the largest numerator term in a cross-product computation, the largest tableau denominator and the largest coefficient numerator found in any tableau obtained while solving the last problem, in that order. All numbers are expressed in base 10. ■ The result is meaningful even if the computation was aborted. ■ See page 8.

`totalValueStats[`*serverId*`]`

`totalValueStats[`*serverId*`]` returns the information on the largest coefficient and subterm values reached while trating problems solved so far on server *serverId*.

■ If the server identification is omitted, 'defaultPip' is assumed. ■ 'totalValueStats' returns a list of `String` objects representing the textual form of the absolute values of the largest numerator term in a cross-product computation, the largest tableau denominator and the largest coefficient numerator found in any tableau obtained while solving the last problem, in that order. All numbers are expressed in base 10. ▪ The result takes into account all computations, including aborted ones. ▪ See page 8.

4.6.5 Problem Conversion

`minToMaxPb[`*pb*`]`

`minToMaxPb[`*pb*`]` converts the problem *pb* given in minimization form to a maximization problem over the same polyhedron.

▪ If the problem is already in maximization form, '`minToMaxPb`' issues a warning message and returns the problem unmodified. ▪ See page 10.

`maxToMinPb[`*pb*`]`

`maxToMinPb[`*pb*`]` converts the problem *pb* given in maximization form to a minimization problem over the same polyhedron.

▪ If the problem is already in minimization form, '`maxToMinPb`' issues a warning message and returns the problem unmodified. ▪ See page 10.

`nextMinPb[`*pb*`]`

`nextMinPb[`*pb*`]` converts a problem determining the lexicographic minimum of $(x_1 \ldots x_n)$ as function of $(z_1 \ldots z_p)$ into a problem determining the lexicographic minimum of $(x_2 \ldots x_n)$ as function of $(z_1 \ldots z_p, x_1)$ under the same set of constraints on $(x_1 \ldots x_n, z_1 \ldots z_p)$ and $(z_1 \ldots z_p)$.

▪ If the variable vector is reduced to one element, '`nextMinPb`' displays a warning message and returns the symbol '`System‘$Failed`'. ▪ See page 10.

`prevMinPb[`*pb*`]`

`prevMinPb[`*pb*`]` converts a problem determining the lexicographic minimum of $(x_1 \ldots x_n)$ as function of $(z_1 \ldots z_p)$ into a problem determining the lexicographic minimum of $(z_p x_1 \ldots x_n)$ as function of $(z_1 \ldots z_{p-1})$ under the same set of constraints on $(x_1 \ldots x_n z_1 \ldots z_p)$ and $(z_1 \ldots z_p)$.

▪ If the parameter vector is empty, '`nextMinPb`' returns a warning message and returns the symbol '`System‘$Failed`'. ▪ See page 10.

5 Limitations, Bugs, Comments and Improvements

5.1 Limitations

By default, the integers exchanged over MathLink are represented on 32 bits. Larger numbers require textual representation, implying extra coding and decoding. Therefore, coefficients with absolute values above the $\pm 2^{31}$ limit are not handled in either direction (neither input nor output).

The conversion functions '`nextMinPb`' and '`prevMinPb`' could be wiser and move elementary constraints between the main constraint system and the context whenever appropriate; e.g., '`nextMinPb`' could move all constraints with variable coefficients identically null to the context, whereas '`prevMinPb`' could move to the main constraint system any context constraints with coefficients non-null for the parameter being removed. However, a systematic application of this rules gives no control on the redundancy of constraints nor the alterations of the context, and can dramatically modify the performance of the server.

5.2 Bugs, Comments and Improvements

Send all your bug reports, comments and improvement suggestions/patches by e-mail to

`<chamskiz@cs.man.ac.uk>`

or, alternatively, by surface mail to

Dr. Zbigniew Chamski
Centre for Novel Computing
Department of Computer Science
University of Manchester
Oxford Road
Manchester M13 9PL
United Kingdom

References

[1] Z. Chamski. *Environnement logiciel de programmation d'un accélérateur de calcul parallèle.* PhD thesis, Université de Rennes I, February 1993.

[2] Z. Chamski. Fast and efficient generation of loop bounds. In D. Trystram, editor, *ParCo '93*, Grenoble, France, September 1993. Elsevier Science Publishers (North Holland). To appear. Available as Inria Research Report 2095, Inria, Rocquencourt, France, October 1993.

[3] Z. Chamski. Nested loop sequences: Towards efficient loop structures in automatic parallelization. In *27th Hawaii International Conference on System Sciences*, volume II: Software Technology, pages 14–22, 1994.

[4] N.V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of a system of linear equations. *USSR Computational Mathematics and Mathematical Physics*, 4(4):151–158, 1964.

[5] J.-F. Collard, P. Feautrier, and T. Risset. Construction of DO loops from systems of affine constraints. Research Report 93-15, École Normale Supérieure de Lyon, Lyon, France, May 1993.

[6] P. Feautrier. Parametric integer programming. *Recherche Opérationnelle / Operations Research*, 22(3):243–268, 1988.

[7] P. Feautrier. Semantical analysis and mathematical programming. Application to parallelization and vectorization. In M. Cosnard et al., editors, *Parallel and Distributed Algorithms*, pages 309–320. Elsevier Science Publishers B. V. (North-Holland), 1989.

[8] P. Feautrier and N. Tawbi. Résolution de systèmes d'inéquations linéaires ; mode d'emploi du logiciel PIP. Technical Report 90.2, Laboratoire MASI, Univ. Pierre et Marie Curie, Paris, France, January 1990.

[9] P. Feautrier and M. R. Werth. Systematic construction of programs for distributed memory computers. Technical Report 91.36, Laboratoire MASI, Univ. Pierre et Marie Curie, Paris, France, June 1991.

[10] T. Granlund. GNU MP: The GNU multiple precision arithmetic library. Reference manual, Free Software Foundation, Inc., December 1991.

[11] H. Le Verge. A note on Chernikova's algorithm. Publication interne 635, Irisa, Rennes, France, February 1992.

[12] Ch. Mauras. *Alpha : un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones.* PhD thesis, Univ. Rennes I, Rennes, France, December 1989.

[13] Wolfram Research. MathLink reference guide. Technical Report 12/92, Wolfram Research, Inc., 1992.

[14] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley Interscience Series in Discrete Mathematics. John Wiley and Sons, 1986.

[15] Stephen Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley, Reading, Mass., USA, 2nd edition, 1991.

Index