

BAX Project

(Bridge between C and Java applications by means of XML)

Project members:

Mladen Vavić (0324418) mvavic@liacs.nl

Jonathan Guzman Carmona (0372188) jguzmanc@liacs.nl

Supervisor: Bart Kienhuis kienhuis@liacs.nl

Date: 08 October 2006

Faculty: Leiden Institute of Advance Computer Science
Faculty of Informatics
Leiden University

Version: 2.0

Abstract

This report discusses the BAX (Bachelor and XML) project results. The main aim of the project was to implement a bridge by means of XML between C libraries and Java API's used by the Compaan tool. The reason for the implementation was the copyright issues posed by the JNI (Java Native Interface) as the bridge between the C libraries and their corresponding Java API's. The result of this project is the implementation of a bridge for each, the Piplib and Polylib libraries, between their corresponding Java API's. The efforts made by this project resulted in the idea of a possible continuation of this project by extending the implemented front-ends on the C and Java software into web services making use of SOAP (Simple Object Access Protocol). This extension would contribute to the efficiency and portability of the Compaan tool because the C libraries would not be required to reside on the same machine.

Acknowledgements

The project group members would like to thank, the project group supervisor, Dr.ir. Bart (A.C.J.) Kienhuis for his guidance, assistance and practical advices during the whole execution of this project.

Contents

1.	Context.....	5
2.	Problem Description.....	6
3.	Project Plan.....	8
4.	Results.....	11
4.1	Piplib.....	11
4.2	Polylib.....	13
4.2.1	The Polylib Interface.....	13
4.2.2	The ZPolylib Interface.....	20
4.3	Correctness.....	21
5	Difficulties.....	22
6	Conclusion.....	23
	Appendices.....	24
1.	How to install and use the libraries.....	24
1.1	Piplib.....	24
1.2	Polylib.....	26
2	Correctness Tests.....	28
2.1	Piplib correctness tests.....	28
2.2	Polylib correctness tests.....	38
3	Literature List.....	56
4	Word list.....	57
5	UML Drawings.....	59

1. Context

The Compaan project is an effort to automatically compile a subset of imperative programs into a concurrent representation. The Compaan tool uses Matlab language as the imperative language and compiles programs in this language into a concurrent representation: a particular version of Process Networks.

In order to achieve this goal, Compaan uses among others, libraries written in C to manipulate and perform calculations on data. Two of these libraries are Piplib and Polylib. Since most of the source code for the Compaan project has been written in Java, JNI (Java Native Interface) has been used to set a bridge between the C libraries and the Java programs. The problem with this approach is that, since the Compaan project has been brought under the CPL license, there exist copyright issues. These issues arose because the license under which JNI has been brought doesn't match the CPL license scheme. This issue led to the concept of implementing a bridge between the C libraries and the Java programs without having to mind license incompatibility problems. The answer to this problem was the use of XML as a bridge between C and Java programs, which will be explained thoroughly in the next section.

2. Problem Description

As already described in the Context section, the use of JNI as a bridge between the C libraries and Java programs posed copyright issues between the different license schemes under which each JNI and the Compaan project had been brought. Therefore a bridge needed to be implemented without the copyright issues.

The C libraries that needed to make use of this new bridge were Piplib and Polylib. Piplib is a tool used to solve Parametric Integer Programming Problems. Piplib finds the lexicographic minimum of the set of integer points which lie inside a convex polyhedron, when that polyhedron depends linearly on one or more general parameters. Polylib is a library that operates on objects like vectors, matrices, lattices, polyhedra, Z-polyhedra, union of polyhedra and other intermediary structures. The Polylib library provides functions for all important operations on these structures. Polylib was developed while working on parallelization techniques and therefore its main users belong to this community.

JNI stands for Java Native Interface and it is a technology developed by Sun to integrate programs written in the Java language with existing non-Java language services, API toolkits, and programs. The JNI defines a standard naming and calling convention so the Java virtual machine can locate and invoke native methods.

The next figure depicts how the libraries were used when JNI was used as a bridge:

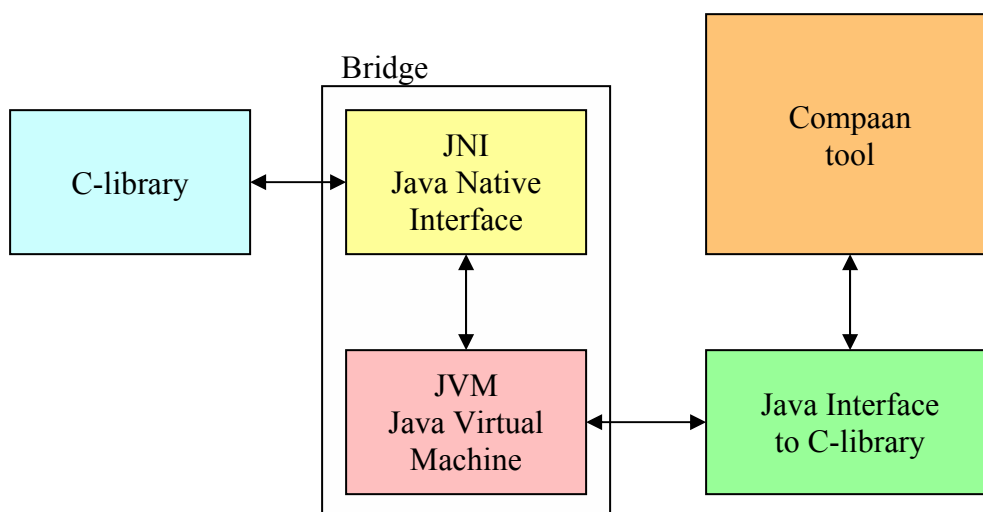


Figure 1

The Compaan tool made use of an API, which was an interface to the C-library, mapping the C functions onto Java calls. This was done by defining an interface in Java and implementing the mapping in .h and .c files where the Java data structures were converted to the C data structures and vice versa and calls were made to the functionality provided by the libraries. The Java virtual machine made then use of its built in JNI capability to locate and invoke the native methods in the C library.

In order to avoid the copyright issues between the different license schemes, the current bridge needed to be replaced. The concept was to replace the use of JNI by letting the C-libraries and Java API's communicate with each other by means of XML messages. It was the challenge for this project to design and implement a conceptual bridge that made use of XML for the Piplib- and Polylib-library. The design and implementation of this bridge will be explained during the next sections.

3. Project Plan

In order to guide this project into a successful completion, a project plan had to be made. This section discusses how the project group members planned to work during the project and how finally the plan was executed.

The project main aim was to implement a bridge between the in C written Piplib- and Polylib-libraries and the Java API used in the Compaan tool by means of XML. Since Piplib and Polylib are two distinct libraries, it was advisable to start first with one library and after completion with the second one. The project counselor gave us the advice to start first with the Piplib-library, since it was the easiest to implement the bridge for. Furthermore it was clear that the implementation of a bridge for each library could only be realized by implementing a module to parse XML data at the front-end of the C-library and the front-end of the Java API for the corresponding library. Also a module needed to be implemented at the front-end of the C-library and the Java API for the corresponding library to produce XML data in order to communicate with each other. This concept is depicted in the figure below:

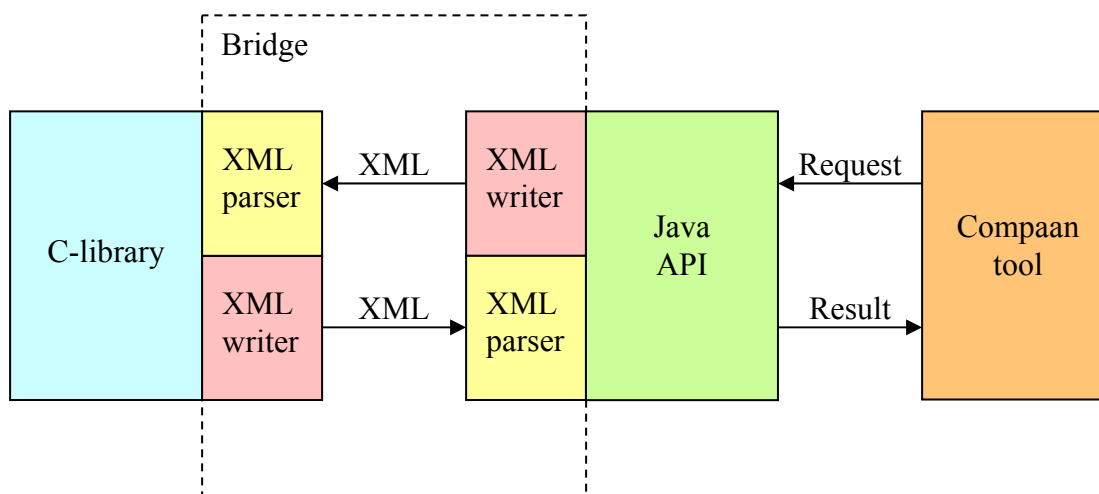


Figure 2

At this point the project plan looked as follows:

Milestone	Task Description
	Learn about XML and how it could be used for the project.
	Learn about Expat for the parsing of XML at the C side.
	Read the documentation of Piplib in order to learn what the library does, what its input- and output data is.
	Design XML protocols (tags and format) for the input data of Piplib.
	Design XML protocols (tags and format) for the output data of Piplib.
	Implement the module to produce XML data (output of Piplib).
	Implement the module to parse the incoming XML data (input for Piplib), use an example XML file as input for testing.
Milestone 1	Completion of the front-end for the Piplib library.
	Start learning about JAXP (Java API for XML Processing).
	Design a Java API to use the Piplib library.
	Implement the module to produce XML data (output from the Java side).
	Write JUnit tests to use the library, which will facilitate the implementation for the parsing of the XML input (output of Piplib).
	Implement the module to parse the incoming XML data.
	Write JUnit tests to test the library.
Milestone 2	Completion of the front-end for the Java API of Piplib.

After completion of the Piplib library, a start needed to be made for the second library. Since the project group members gained experience with XML, Expat and JAXP during the bridge implementation of the first library, it was easier to focus on the implementation of the bridge for the Polylib library. The Polylib library consists of two interfaces for which functions needed to be implemented. These two interfaces are PolyLib and ZPolylib. The following is the planning for the Polylib library:

Milestone	Task Description
	Read the documentation of Polylib in order to learn what the library does, what its input- and output-data is.
	Design XML protocols (tags and format) for the input data of Polylib.
	Design XML protocols (tags and format) for the output data of Polylib.
	Create a module that can convert a Java data structures into XML format.
	Create a module that can convert an XML representation of a data structure into a Java object. For this, a parser is needed and a content handler. Make some skeleton-code for a content handler and implement it along with the implementation of functions in the Polylib and ZPolylib interfaces.
	Start implementing the PolyLib interface first. For this a module is required to convert C data structures into XML format at the front-end of the Polylib library and a module to parse incoming XML data from the

	Java side at the front-end of the Polylib library.
	Write JUnit tests for each function that has been implemented.
Milestone 3	Completion of the bridge for the PolyLib interface.
	After completion of the PolyLib interface implement the functions of the ZPolyLib interface by extending the existing modules to create and parse XML data on both front-ends.
	Write JUnit tests for each function that has been implemented.
Milestone 4	Completion of the bridge for the ZPolylib interface.
Milestone 5	Write documentation for both the Piplib library and the Polylib library.

Finally, the bridges between the Piplib- and Polylib-libraries with its correspondent Java API's were successfully implemented by executing the described project plan. The project plan could not have been successfully executed without good communication between the project's members and discussion sessions with the project's counselor between and after completion of each milestone.

4. Results

This section discusses the results produced during this project. These results concern the implementation of a bridge between the in C written libraries Piplib and Polylib and their corresponding Java API by means of XML communication. Furthermore at the end of this section, the correctness of the new implemented bridge is discussed.

4.1 Piplib

The Piplib library stands for parametric integer linear solver library. The library is a piece of software used to find the lexicographic minimum of the set of integer points lying inside a convex polyhedron. This means that this library is intended for a single job and therefore it has a well defined input and output format.

The Piplib takes as input only six parameters and two matrices. The first four parameters are the number of unknowns, number of symbolic parameters, number of inequalities and number of inequalities satisfied by parameters. The remaining two parameters are the number of parameters + 1 or 1 and 0 for integer valued solution or nonzero for lexicographic. These parameters are followed two matrices carrying information about the domain and context. Each matrix consists of certain number of vectors which can be seen as one row of matrix. The XML input for this library is listed as follows:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<pip>
  <parameter name="Nn" value="#" />
  <parameter name="Np" value="#" />
  <parameter name="Nl" value="#" />
  <parameter name="Nm" value="#" />
  <parameter name="Bg" value="#" />
  <parameter name="Nq" value="#" />
  <tableau>
    <vector number="#">
      <element column="#" value="#" />
      ...
    </vector>
    ...
  </tableau>
  <context>
    <vector number="#">
      <element column="#" value="#" />
      ...
    </vector>
    ...
  </context>
</pip>
```

The XML output produced by the library is listed below:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE pip SYSTEM "dtd/pipsolution.dtd">
<pip>
  <solution>
    <if>
      ...
      <then>
        ...
      </then>
      <else>
        ...
      </else>
    </if>
  </solution>
</pip>
```

The Piplib library is implemented as follows: a PipSolver class defines an API with functions to initialize the object, set a problem, set a context and solve to execute the C library front-end on the command line and find a solution. The PipSolver object creates on its own XML input data for the C library front-end. The C library front-end uses a module (pipXMLInput) together with Expat (C SAX library for parsing XML) to parse the incoming XML data with definition of the problem to be solved and a module (pipXMLOutput) to convert the solution into a XML representation. Both modules are controlled by a pipXML program which is the one called on the command line by the PipSolver object on the Java front-end.

When the PipSolver object receives the XML output data from the C front-end, it parses the XML data with the solution to the problem. This parsing is done by using the Xerces parser, which in turn uses a content handler (PipContentHandler) that process the data as it is being parsed. This processing results into a parse tree interpretation of the solution.

The parse tree is constructed by using a ParseNode object that forms the basis for the parse tree. This object contains a statement field, which is an object that represents the different kind of statements that can be found in the solution scheme: statement, IfStatement, ListStatement and NewParmStatement.

4.2 Polylib

This section discusses the implementation of the bridge between the Polylib library and its corresponding Java API for further used by the Compaan tool.

The Polylib library, as already discussed, provides a set of functions for all important operations on data structures. The library can be divided into two set of functions for which interfaces can be implemented in Java. The first interface is the Polylib interface and the second one the ZPolylib interface.

4.2.1 The Polylib Interface

The Polylib interface consists of twenty functions that perform operations listed as follows:

Constraints2Polyhedron (SignedMatrix): Polyhedron
 ConstraintsSimplify (SignedMatrix domain, SignedMatrix context): SignedMatrix
 DomainConvex (Polyhedron): Polyhedron
 DomainCopy (Polyhedron): Polyhedron
 DomainDifference (Polyhedron, Polyhedron): Polyhedron
 DomainPreimage (Polyhedron, JMatrix): Polyhedron
 DomainUnion (Polyhedron, Polyhedron): Polyhedron
 EmptyPolyhedron (int): Polyhedron
 Polyhedron2Constraints (Polyhedron): SignedMatrix
 Polyhedron2ParamDomain (Polyhedron, Polyhedron): ParamPolyhedron
 Polyhedron2ParamVertices (Polyhedron, Polyhedron): ParamPolyhedron
 PolyhedronEnumerate (Polyhedron, Polyhedron): Enumeration
 LexSmallerEnumerate (SignedMatrix, SignedMatrix, int, SignedMatrix): Enumeration
 PolyhedronScan (Polyhedron, Polyhedron): Polyhedron
 Rays2Polyhedron (SignedMatrix): Polyhedron
 UniversePolyhedron (int): Polyhedron
 DomainImage (Polyhedron d, JMatrix m): Polyhedron
 DomainIntersection (Polyhedron d1, Polyhedron d2): Polyhedron
 PolyhedronCount (Polyhedron, EvaluateFunction): void
 DomainSimplify (Polyhedron, Polyhedron): Polyhedron

From the functions listed above, it was not a requirement to implement the function PolyhedronCount. The following is a listing of the data structures required by this interface each in its XML representation and C definition. The corresponding Java definitions are the same as C, but in classes with private variables, public getters and setters for accessing these variables and constructors for creating an instance of the equivalent Java object.

JMatrix:

XML representation

```
<JMatrix nrRows="#" nrCols="#">
  <element row="#" col="#" value="#" />
  ...
</JMatrix>
```

C definition

Matrix	struct matrix	
	unsigned	NbRows
		NbColumns
	Value	**p
*p_Init		

SignedMatrix:

XML representation

```
<signedMatrix nrRows="#" nrCols="#">
  <element row="#" col="#" value="#" />
  ...
</signedMatrix>
```

C definition

Matrix	struct matrix	
	unsigned	NbRows
		NbColumns
	Value	**p
*p_Init		

Polyhedron:

XML representation

```
<polyhedrons>
  <polyhedron>
    <dimension value="#" />
    <nrConstraints value="#" />
    <nrRays value="#" />
    <nrEqualities value="#" />
    <nrBid value="#" />
    <constraints nbRows="#"
      nbCols="#">
      <constraint row="#"
        col="#" />
      ...
    </constraints>
    <rays>
      <ray row="#" col="#"
        value="#" />
      ...
    </rays>
  </polyhedron>
  ...
</polyhedrons>
```

C definition

Polyhedron	struct polyhedron	
	unsigned	Dimension
		NbConstraints
		NbRays
		NbEq
		NbBid
	Vector	**Constraint
**Ray		
struct polyhedron	*p_Init	
		*next

ParamDomain:**XML representation**

```

<paramDomains>
  <paramDomain>
    <polyhedrons>
      ...
    </polyhedrons>
    <booleanVector length="#">
      <boolean index="#"
        value="#" />
      ...
    </booleanVector>
  </paramDomain>
  ...
</paramDomains>

```

ParamVertices:**XML representation**

```

<paramVertices>
  <paramVertice>
    <domain>
      <signedMatrix nrRows="#"
        nrCols="#">
        ...
      </signedMatrix>
    </domain>
    <vertex>
      <signedMatrix nrRows="#"
        nrCols="#">
        ...
      </signedMatrix>
    </vertex>
  </paramVertice>
  ...
</paramVertices>

```

C Definition

Param_Domain	struct Param_Domain	
	unsigned	*F
	Polyhedron	*Domain
	struct Param_Domain	*next

C definition

Param_Vertices	struct Param_V	
	struct Param_V	*next
	Matrix	*Vertex
		*Domain

ParamPolyhedron:**XML representation**

```

<paramPolyhedron>
  <nbV value="#" />
  <paramDomains>
    ...
  </paramDomains>
  <paramVertices>
    ...
  </paramVertices>
</paramPolyhedron>

```

C definition

Param_Polyhedron	struct Param_Poly	
	int	nbV
	Param_Vertices	*V
	Param_Domain	*D

ENode:**XML representation**

```

<eNode>
  <size value="#" />
  <position value="#" />
  it can be a:
  <etype value="evector" />
  <eVector>
    <eValue>
      ...
    </eValue>
  </eVector>
  or it can be a:
  <etype value="polynomial" />
  <polynomial>
    <eValue>
      ...
    </eValue>
  </polynomial>
  or it can be a:
  <etype value="periodic" />
  <periodic>
    <eValue>
      ...
    </eValue>
  </periodic>
</eNode>

```

C definition

enode	struct enode	
	enode_type type	type
	int	size
	evaluate	pos
		arr[1]

EValue:**XML representation**

```

<eValue>
  it can be a:
  <fraction numerator="#" \
              denominator="#" />
  or it can be a:
  <nValue value="#" />
  or it can be a:
  <eNode>
  ...
  </eNode>
</eValue>

```

C Definition

evaluate	struct_evalue	
	Value	*ValidityDomain
	Value	n
	struct_enode	*p

Enumeration:**XML representation**

```

<enumerations>
  <enumeration>
    <polyhedrons>
    ...
    </polyhedrons>
    <eValue>
    ...
    </eValue>
  </enumeration>
  ...
</enumerations>

```

C Definition

Enumeration	struct_enumeration	
	Polyhedron	*ValidityDomain
	evaluate	EP
	struct_enumeration	*next

Having listed the data structures used by the PolyLib interface, this is how the XML input looks like for a call to a function on the C library, generated on the Java front-end:

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<polylib>
  <function name="name of the function" nrParams="#">
    <param nr="#">
    ...
    </param>
    ...
  </function>
</polylib>

```

The tag function has an attribute name which identifies the function to be executed and the number of parameters that need to be passed for the corresponding function. The function tag is followed by one or more param-tags that enclose the data structures that need to be passed. These data structures can be any of the previously discussed structures. Furthermore each param-tag has an attribute number which determines the parameter order by which the data structures are passed.

The Java API for the PolyLib interface works as follows: an instance of the PolyLib object is instantiated. The desired function is called by passing the right parameters. The PolyLib interface instantiates a PolylibXMLImplementation object and delegates the function call to this object. This object in turn creates the XML input for the C Polylib's front-end. This input is created by means of the XMLInputCreator object, which in turn uses the PolyTypes2XML object to convert the data structures into XML format that need to be passed to the function. The PolylibXMLImplementation object executes the C implementation of the Polylib library that can handle XML input data. The execution is done by means of the PolylibExecuter object. This object executes the Polylib library which parses the XML input by converting the XML formatted data structures into their C representations, executes the requested function which returns some data structure. This data structure is then converted again into its XML representation and this result sent back to the PolylibExecuter object. The PolylibExecuter object receives the XML input and converts the XML formatted data structure into a correspondent Java object. The conversion is done by using the XML2PolyTypes object which in turn uses an XML parser which in turn uses a content handler to parse the XML data. This content handler is a custom object PolylibContentHandler used by the parser to do the conversion of the XML data as is being processed. Figure 3 depicts the concept discussed in this paragraph for the Java front-end.

The front end of the Polylib library is implemented by two modules. The first module is polylibXMLInput, which parses the XML input together with Expat, converts the XML formatted data structures into C structures and executes the requested functions on the passed parameters. The second module is polylibXMLOutput which covers the resulted C structure into XML format. Both modules are controlled by a polylibXML program which is the one called on the command line by the PolylibExecuter object on the Java front-end. This concept is depicted in figure 4.

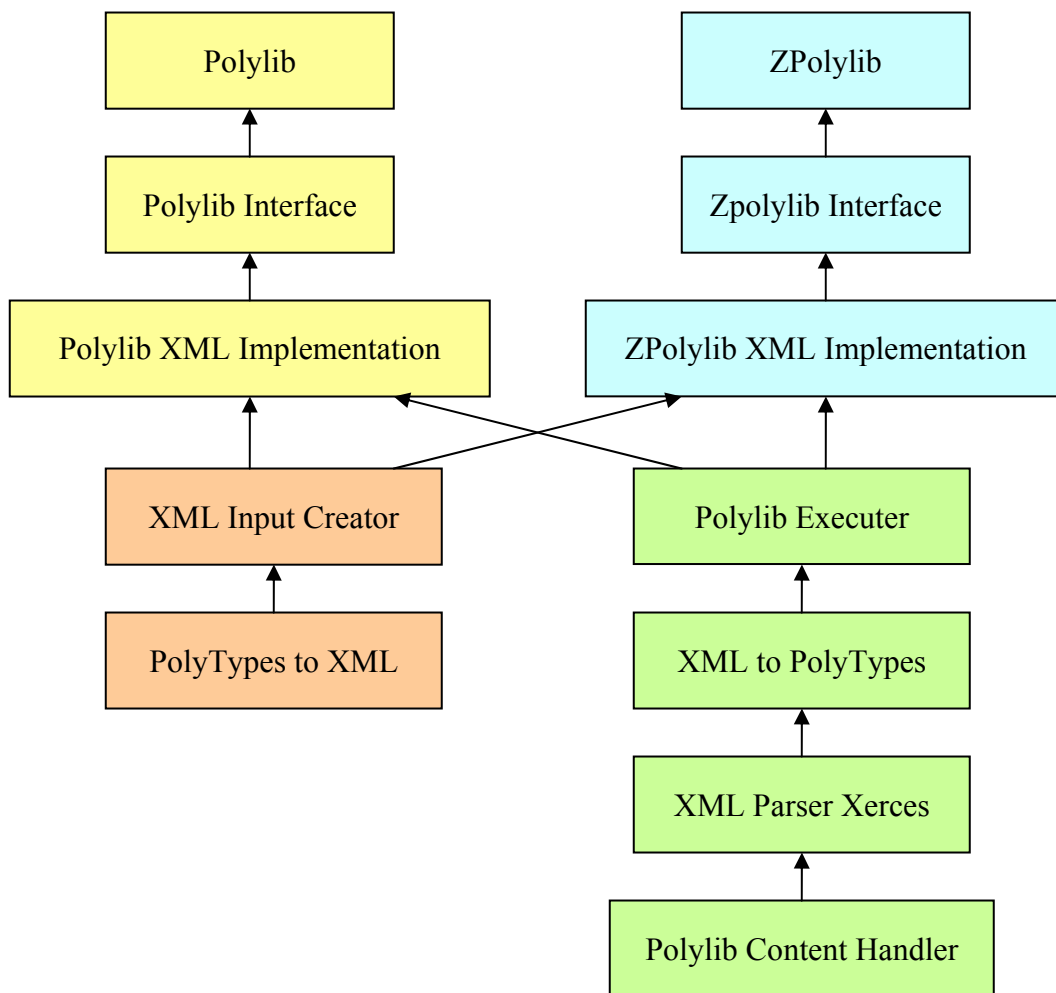


Figure 3

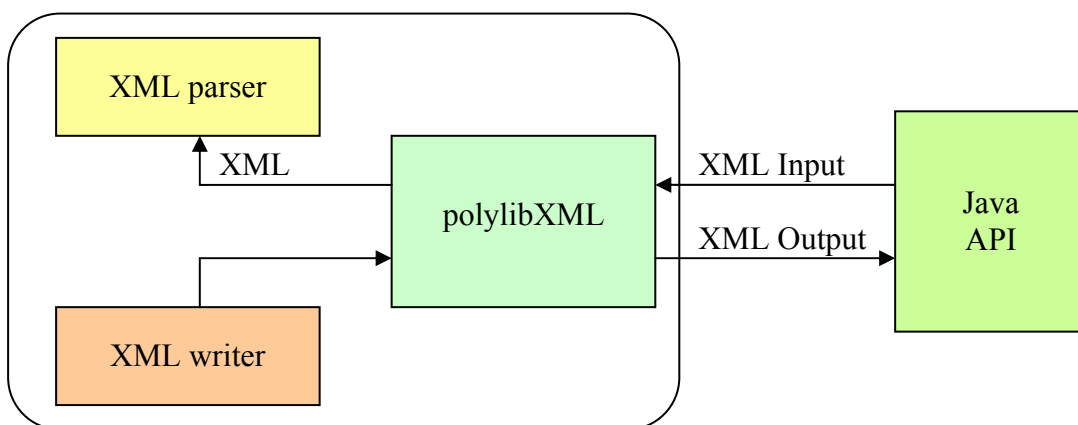


Figure 4

4.2.2 The ZPolylib Interface

The ZPolylib interface consists of twenty two functions that perform operations on data structures. Only six functions are used by the Compaan tool. These functions are the following:

```
affineHermite (Lattice): Lattice
getHermite (SignedMatrix): SignedMatrix
getSmithNormalProduct (SignedMatrix): SignedMatrix
getSmithNormalU (SignedMatrix): SignedMatrix
getSmithNormalV (SignedMatrix): SignedMatrix
getUniModular (SignedMatrix): SignedMatrix
```

The following is a listing of the data structures required by this interface as far as it is concerned for the functions listed above. The listing consists of the XML representation and C definition. The corresponding Java definition is the same as C, but in classes with private variables, public getter and setters for accessing these variables and constructors for creating an instance of the equivalent Java object.

SignedMatrix:

See SignedMatrix in previous section.

Lattice:

XML representation

```
<lattice nrRows="#" nrCols="#">
  <element row="#" col="#"
    value="#" />
  ...
</lattice>
```

C definition

```
Lattice typedef Matrix
```

The implementation of this interface was done by extending the Java objects discussed in the previous section: XML2PolyTypes, XMLInputCreator, PolylibExecuter, XML2PolyTypes and PolylibContentHandler on the Java front-end. The C front-end needed to be extended as well to cope with the data structures and functions involved with this interface. The Zpolylib interface works as described in the previous section for the Polylib interface only that the XML input for the C front-end contains requests for the functions implemented by this library and the XMLresponse contains the XML representation of the data structures returned by the requested functions.

4.3 Correctness

In order to verify the correctness of the implementation of the new bridge for each library, it was imperative to run tests that would confirm that indeed the parsing of XML was done properly on both C and Java front-ends.

It was clear that writing only JUnit tests for each Java API would not be the solution. The reason for this is that by using only assertions, it was not possible to test the contents of the data structures returned by the Java API properly. Therefore, the project group set up a test framework with C programmes resembling each JUnit test in Java. These C programmes make use of the original libraries without the XML front-end while the JUnit tests do make use of the libraries with the XML front-end. Both C programmes and JUnit tests make use of the exact same input. This way the output of each C program and its corresponding JUnit test would be compared for equalness, proving the implementation to be correct.

The output of the C programmes and their corresponding JUnit tests concern the printing of the contents of the resulting data structure after applying some operation on the library. Therefore some printing functions needed to be implemented in C and Java so the comparison of the tests' output would be easier. Since Polylib doesn't provide functions to print the contents of Enumerations and ParamPolyhedrons, functions for printing the contents of these data structures were implemented for the testing framework. Also some methods needed to be implemented on the Java side in order to print the contents of these data structures in a similar way.

After running the C programmes and the JUnit tests, each output in C and Java was carefully compared. This led to the conclusion that the bridge implemented for the Piplib and Polylib library was correct; since the parsing on both front-ends was done properly (namely both outputs were the same). The project group refers to the Correctness Tests section in the appendice for detailed information on the tests.

5 Difficulties

During the execution of the project plan, there were not really many difficulties faced by the project group members. The project group managed to handle all of them nicely.

The difficulties faced were the following:

- Poorly documented source code for the C libraries.
- Poorly documentation on the libraries themselves.
- Lack of theoretical mathematical knowledge used by the libraries.
- Expat lack of validation of XML content against DTD validation files.
- Problems to test the implementations with assertions in JUnit tests.

6 Conclusion

The project was successfully completed by the project group members. The project group members managed to implement the bridges between the Piplib- and Polylib libraries with their corresponding Java API's by transferring XML files between both the Java and C front-ends. This implementation resolved the copyright issues posed by the former integration of the C libraries into the Compaan tool.

The implementation was finally tested by writing and executing JUnit tests for each function implemented in each library. Since it was difficult to verify the results of each implemented function by assertions, the project group members chose to set up a test framework in which each JUnit test has its corresponding test written in C with identical input. These tests made use of the libraries without the XML front-end while the JUnit tests made use of the libraries with the XML front-end. The output of the C tests and the JUnit tests were compared to be equal, which proved the new implementation of the bridge to be correct.

As a result of the efforts made during this project, this project can be extended to the implementation of web services. Since the Java API's communicate with the in C written libraries by means of XML, it is not required to have both front-ends residing on the same location. Because of the portability of XML, one could place the C libraries with their implementation of XML front-ends on a different location and have the Java API's request the service of the library through the internet by means of SOAP (Simple Access Object Protocol). This way the computations done by the C libraries are done in other locations, minimizing the execution time and contributing to the efficiency of the Compaan tool.

Appendices

1. How to install and use the libraries

1.1 Piplib

The Piplib library consists of a C and a Java source code, where each front-end of the bridge is implemented. Both source codes need to be compiled in order to use the library.

Compiling and installing Piplib's C front-end

The C source code is available through CVS at *cvs.liacs.nl* and it is located at */cvs/lerc/projects/bax/Piplib-1.3.6*. The front-end of the Piplib library is based on Piplib version 1.3.6 and requires this library to be installed first in order to be compiled.

The Piplib library can be installed by entering into the Piplib-1.3.6 directory, type *configure* and the *make*, which will compile the package and install the program and the library. By default, the library is installed in */usr/local/bin*. The location of the library can be adjusted. For more details, please consult Piplib's manual at http://www.prism.uvsq.fr/~cedb/bastools/pip_manual/index.html.

After Piplib has been installed, the source code for the C front-end can be compiled, provided that Expat is installed on the system. For more information on Expat, please consult its documentation on <http://expat.sourceforge.net/>. During this project expat-2.0.0 was used. The source code for the C front-end can be found at */cvs/lerc/projects/bax/piplib-1.3.6/piplibXml*. In order to compile the source code to the *piplibXml* directory type *make*. This will create the executable *piplibXML#* (*# is the version of the library*) used by the Java API.

The makefile used to compile the contents of the *piplibXML* directory uses the variable *LIBSDIR = -L/usr/local/lib*. This variable is passed to the *gcc* compiler to find the location of the Expat and Piplib libraries. If this location is different on the system to be installed, then change this variable to the right location.

Uninstalling Piplib's C front-end

In order to uninstall these libraries, type *make clean* inside the *piplibXML* directory then go one level up and type *make uninstall*. This will uninstall the Piplib library and the executable created inside the *piplibXML* directory.

Compiling Piplib's Java front-end

The Java source code is available through CVS at *cvs.liacs.nl* and it is located at */cvs/lerc/projects/kernel*. To compile the java source code it is required to have the JDK version 5.0 or higher installed on the machine. In order to compile go to the */cvs/lerc/projects/kernel/src* directory, type `configure --with-java="location"` (location is where the installation of the JDK can be found) and then type `make`. To remove the .class files generated by the make command, type `make clean`.

Using the Java Piplib interface

The Piplib library is available through a *PipSolver* object that needs to be instantiated and requires to import the *kernel.solver.pipsolver* package in the Java source code. The next is a listing of how the library can be used:

```
// Creates and initializes a PipSolver
PipSolver piplib = new PipSolve ();
piplib.initialize (9, 6, 24, 1, 10, 0);

// Sets the symbolic parameters to be decoded
Vector<String> decode = new Vector<String>();
decode.add ("k");
decode.add ("m");
decode.add ("n");
decode.add ("o");
decode.add ("p");
decode.add ("q");
piplib.setDecodeString (decode);

// Sets the problem and context to be solve
piplib.setProblem (problem);
piplib.setContext (context);

// Solves the problem and gets a reference to the top Node of
// the data structure containing the solution
ParserNodeImp result = piplib.solve ();
```

Here are *problem* and *context* two *SignedMatrix* objects containing the corresponding problem- and context definitions. After execution of the library, a reference of a *ParseNodeImp* object is returned. This object is a reference to the root node of a parse tree representation of the solution to the problem.

1.2 Polylib

The Polylib library also consists of a C and a Java source code. Both source codes need to be compiled in order to use the library.

Compiling and installing Polylib's C front-end

The C source code is available through CVS at *cvs.liacs.nl* and it is located at */cvs/lerc/projects/bax/polylib-5.22.0*. The front-end of the Polylib library is based on Polylib version 5.22.0 and requires this library to be installed first in order to be compiled.

The Polylib library can be installed by entering into the Polylib-5.22.0 directory. Type *configure* and then *make* and finally *make install*, which will compile the package and install the program and the library. In case of compiling on cygwin, configure command requires extra parameter and should be run as *configure --enable-int-lib*. By default, the library is installed in */usr/local/bin*. The location of the library can be adjusted. For more details, please consult Polylib's manual at <http://www.irisa.fr/cosi/polylib/user/DOC/index.html>.

After Polylib has been installed, the source code for the C front-end can be compiled, provided that Expat (used version is 2.0.0) is installed on the system. The source code for the C front-end can be found at */cvs/lerc/projects/bax/polylib-5.22.0/polylibXml*. In order to compile the source code to the *polylibXml* directory type *make*. This will create the executable *polylibXML#* (*# is the version of the library*) used by the Java API.

The makefile used to compile the contents of the *polylibXML* directory uses the variable *LIBSDIR = -L/usr/local/lib*. This variable is passed to the *gcc* compiler to find the location of the Expat and Polylib libraries. If this location is different on the system to be installed, then change this variable to the right location.

Uninstalling Polylib's C front-end

In order to uninstall these libraries, type *make clean* inside the *polylibXML* directory then go one level up and type *make uninstall*. This will uninstall the Polylib library and the executable created inside the *polylibXML* directory.

Compiling Polylib's Java front-end

The Java source code is available through CVS at *cvs.liacs.nl* and it is located at */cvs/lerc/projects/kernel*. To compile the java source code it is required to have the JDK version 5.0 or higher installed on the machine. In order to compile go to the */cvs/lerc/projects/kernel/src* directory, type *configure --with-java="location"* (location is where the installation of the JDK can be found) and then type *make*. To remove the .class files generated by the make command, type *make clean*.

Using the Java Polylib interface

The Polylib library is available through a *PolySolver* object that needs to be instantiated and requires to import the *kernel.solver.polysolver* package in the Java source code. The next is a listing of how the library can be used:

```
// Gets an instance of the PolySolver object
PolyLib polySolver = PolyLib.getInstance();

// stores result of a function
Polyhedron p1 = polySolver.Constraints2Polyhedron(problem);
```

Here is problem *SignedMatrix*, an object containing the corresponding problem definition. After execution of *Constraints2Polyhedron* function, *p1* contains a reference to the resulting *Polyhedron*.

2 Correctness Tests

As discussed in section 4.3 Correctness, a test framework has been implemented. This test framework consists of C programmes resembling each JUnit test in Java. The framework is available through CVS at *cvs.liacs.nl* and it is located at */cvs/lerc/projects/bax/testframework*. The tests can be compiled by running the script */compile* on the command line. To run the tests, run the *./run* on the command line. The JUnit tests are integrated in the java source code and thus are available through CVS at *cvs.liacs.nl* and it is located at */cvs/lerc/projects/kernel*. To run the JUnit tests go to *kernel/src* and type *make test*.

2.1 Piplib correctness tests

Two tests were executed to check if output of the original C library was the same as the one from the JUnit test. The following are input- and output-data of C front-end program and JUnit test for each test.

Input data for test 1:

```
#problem
24 17
# eq/in          unknowns          parameters          cst
1  0 0 0 -1 0 0 0 0 0  1 -1 1 0 0 0 0  0
1  0 0 0 0 0 0 0 -1 0 0  1 -1 1 0 0 0 0  0
1  0 0 0 0 -1 0 0 0 0 0  1 -1 1 0 0 0 0  0
1  0 0 0 0 0 0 0 0 -1 0  1 -1 1 0 0 0 0  0
1  0 0 0 0 0 0 1 0 0 0  -1 1 0 -1 0 0 0  0
1  -1 0 0 0 0 0 0 0 0 0  1 -1 0 1 0 0 0  0
1  0 -1 0 0 0 0 0 0 0 0  1 -1 0 0 1 0 0  0
1  0 0 1 0 0 0 0 0 0 0  -1 1 0 0 0 0 -1  -1
1  0 0 0 1 0 0 0 0 0 0  -1 0 0 0 0 0 0  -1
1  0 0 0 1 1 -1 0 0 0 0  -1 0 0 0 0 0 0  0
1  0 0 -1 0 0 0 2 0 0 0  -1 0 0 0 0 0 0  0
1  0 0 0 0 0 0 1 0 0 0  -1 0 0 0 0 0 0  -1
1  0 0 0 0 0 0 1 1 -1  -1 0 0 0 0 0 0  0
1  0 0 0 0 1 0 0 0 0 0  -1 0 0 0 0 0 0  -1
1  0 0 0 0 0 0 1 0 0 0  -1 0 0 0 0 0 0  -1
1  0 0 0 0 0 0 0 0 0 1  -1 0 0 0 0 0 0  -1
1  1 1 0 0 0 0 0 0 0 0  -2 2 0 -1 -1 0 0  0
1  0 0 0 2 0 0 0 0 0 0  -2 1 0 0 0 0 -1  0
1  0 -1 0 0 0 0 0 4 3  -6 0 0 0 0 0 0  0
1  0 0 0 0 4 3 0 0 0 0  -7 1 0 0 -1 0 0  0
1  1 1 1 0 0 0 -2 -4 -4  7 0 0 0 0 0 0  0
1  0 0 0 -2 -4 -4 0 0 0  10 -3 0 1 1 1 0  0
1  -1 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0  0

#context
1 8
# eq/in          cst
1  0 -1 1 0 0 0  -1
```



```

newparm div8= 7 div(m+q+1,2)
newparm div9= 8
div(2*m+3*o+3*p,4)
list <k-m+o, k-m+p,
k+2*m+3*o+3*p+2*q-2*div8-4*div9+2, k-2*m+4*o-div7, k+m-3*o+div7, k-m+o, k+q-div8+1, k-m-3*o-2*p+3*div9,
k+m+4*o+3*p-4*div9>
else
else
else
else
newparm div10= 6 div(m+q,2)
if -m-q+2*div10 >= 0
then
newparm div11= 7
div(3*m+3*o+3*p+q+2*div10,4)
if -6*m-3*o-3*p-4*q+4*div10+4*div11 >= 0
then
newparm div12= 8
div(q+2*div10+2*div11,3)
newparm div13= 9 div(q+div12,2)
newparm div14= 10 div(m+q+1,2)
newparm div15= 11 div(2*m+3*o+3*p,4)
list <k-m+o, k-m+p,
k+2*m+3*o+3*p+2*q-2*div14-4*div15+2, k+q-div10, k+2*m+p+3*q-3*div13, k-3*m-p-4*q+4*div13, k+q-div14+1,
k-m-3*o-2*p+3*div15, k+m+4*o+3*p-4*div15>
else
else
else
else
else
else
if 2*m-3*o+p-4 >= 0
then
if -6*m+4*n+3*o-p >= 0
then
newparm div16= 6 div(m+q,2)
if -m-q+2*div16 >= 0
then
newparm div17= 7 div(3*m+3*o+3*p+q+2*div16,4)
if -6*m-3*o-3*p-4*q+4*div16+4*div17 >= 0
then
newparm div18= 8 div(q+2*div16+2*div17,3)
newparm div19= 9 div(q+div18,2)
newparm div20= 10 div(m+q+1,2)
newparm div21= 11 div(2*m+3*o+3*p,4)
list <k-m+o, k-m+p, k+2*m+3*o+3*p+2*q-2*div20-
4*div21+2, k+q-div16, k+2*m+p+3*q-3*div19, k-3*m-p-4*q+4*div19, k+q-div20+1, k-m-3*o-2*p+3*div21,
k+m+4*o+3*p-4*div21>
else
else
else
else
if 2*m-3*o+p-4 >= 0
then
if -6*m+4*n+3*o-p >= 0
then
newparm div22= 6 div(m+q,2)
if -m-q+2*div22 >= 0
then
newparm div23= 7 div(3*m+3*o+3*p+q+2*div22,4)
if -6*m-3*o-3*p-4*q+4*div22+4*div23 >= 0
then
newparm div24= 8 div(q+2*div22+2*div23,3)
newparm div25= 9 div(q+div24,2)
newparm div26= 10 div(m+q+1,2)
newparm div27= 11 div(2*m+3*o+3*p,4)
list <k-m+o, k-m+p, k+2*m+3*o+3*p+2*q-2*div26-
4*div27+2, k+q-div22, k+2*m+p+3*q-3*div25, k-3*m-p-4*q+4*div25, k+q-div26+1, k-m-3*o-2*p+3*div27,
k+m+4*o+3*p-4*div27>
else
else
else
else
else
if -m+2*n-o >= 0
then

```


Comparison of two pieces out of the outputs:

```

(if #[ 0 -6 0 -3 -3 -4 4 4 0]
  (newparm 8 (div #[ 0 0 0 0 0 1 2 2 0] 3))
  (newparm 9 (div #[ 0 0 0 0 0 1 0 0 1 0] 2))
  (newparm 10 (div #[ 0 1 0 0 0 1 0 0 0 0 1] 2))
  (newparm 11 (div #[ 0 2 0 3 3 0 0 0 0 0 0] 4))
  (list
    #[ 1 -1 0 1 0 0 0 0 0 0 0 0]
    #[ 1 -1 0 0 1 0 0 0 0 0 0 0]
    #[ 1 2 0 3 3 2 0 0 0 0 -2 -4 2]
    #[ 1 0 0 0 0 1 -1 0 0 0 0 0]
    #[ 1 2 0 0 1 3 0 0 0 -3 0 0]
    #[ 1 -3 0 0 -1 -4 0 0 0 4 0 0]
    #[ 1 0 0 0 0 1 0 0 0 0 -1 0 1]
    #[ 1 -1 0 -3 -2 0 0 0 0 0 0 3 0]
    #[ 1 1 0 4 3 0 0 0 0 0 0 -4 0]
  )
)
)
)
)

```

```

if -6*m-3*o-3*p-4*q+4*div1+4*div2 >= 0
then
  newparm div3= 8 div(q+2*div1+2*div2,3)
  newparm div4= 9 div(q+div3,2)
  newparm div5= 10 div(m+q+1,2)
  newparm div6= 11 div(2*m+3*o+3*p,4)
  list <k-m+o, k-m+p, k+2*m+3*o+3*p+2*q-
2*div5-4*div6+2, k+q-div1, k+2*m+p+3*q-
3*div4, k-3*m-p-4*q+4*div4, k+q-div5+1, k-
m-3*o-2*p+3*div6, k+m+4*o+3*p-4*div6>
else

```

If the first if statement is expanded, following inequality is received:

$$0*k + (-6)*m + 0*n + (-3)*o + (-3)*p + (-4)*q + 4*newparm6 + 4*newparm7 + 0 \geq 0 \text{ which is}$$

$$-6*m - 3*o - 3*p - 4*q + 4*newparm6 + 4*newparm7 \geq 0.$$

If newparm 8 is observed, its equation is as:

$$0*k + 0*m + 0*n + 0*o + 0*p + 1*q + 2*newparm6 + 2*newparm7 + 0 \text{ which is}$$

$$q + 2*newparm6 + 2*newparm7.$$

Finally if list is completely expanded, parameters have the following values:

$$1*k + (-1)*m + 0*n + 1*o + 0*p + 0*q + 0*newparm6 + 0*newparm7 + 0*newparm8 + 0*newparm9 + 0*newparm10 + 0*newparm11 + 0, 1*k + (-1)*m + 0*n + 0*o + 1*p + 0*q + 0*newparm6 + 0*newparm7 + 0*newparm8 + 0*newparm9 + 0*newparm10 + 0*newparm11 + 0, \dots, 1*k + 1*m + 0*n + 4*o + 3*p + 0*q + 0*newparm6 + 0*newparm7 + 0*newparm8 + 0*newparm9 + 0*newparm10 + (-4)*newparm11 + 0$$

and that is evaluated simpler as:

$$k - m + o, k - m + p, \dots, k + m + 4*o + 3*p - 4*newparm11.$$

Input data for test 2:

```
#problem definition {-i+m >=0; -j+n >=0; j+i-k >=0}
3 7
#      unknowns      parameters
# eq/in i  j      k  m  n      cst
    1  0 -1      0  1  0      0
    1 -1  0      0  0  1      0
    1  1  1     -1  0  0      0

#context {-k+m+n >=0}
1 5
# eq/in k  m  n  cst
    1 -1  1  1  0
```

C test 2 output:

```
(if #[ -1 1 0 0]
  (list
    #[ 0 0 0 0]
    #[ 1 0 0 0]
  )
  (list
    #[ 1 -1 0 0]
    #[ 0 1 0 0]
  )
)
```

JUnit test 2 output:

```
if -k+m >= 0
  then
    list <0, k>
  else
    list <k-m, m>
```

2.2 Polylib correctness tests

For each function is written one test. Outputs of C front-end test program and JUnit test are as following:

testConstraints2Polyhedron (C front-end test program):

```
POLYHEDRON Dimension:2
          Constraints:4  Equations:0  Rays:4  Lines:0
Constraints 4 4
Inequality: [  1  0  0  ]
Inequality: [ -1  1 -1  ]
Inequality: [  0  1 -2  ]
Inequality: [  0  0  1  ]
Rays 4 4
Ray: [  0  1  ]
Ray: [  1  1  ]
Vertex: [  1  2  ]/1
Vertex: [  0  2  ]/1
```

testConstraints2Polyhedron (JUnit test):

```
Validity Domain(0):
_dimension: 2
_nbConstraints: 4
_nbRays: 4
_nbEq: 0
_nbBid: 0

Inequality: 1 0 0
Inequality: -1 1 -1
Inequality: 0 1 -2
Inequality: 0 0 1

Ray: 0 1
Ray: 1 1
Vertex: 1 2 /1
Vertex: 0 2 /1
```

testConstraintsSimplify (C front-end test program):

```
2 4
 1 -1 1 -1
 1 0 0 1
```

testConstraintsSimplify (JUnit test):

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

testDomainConvex (C front-end test program):

```

POLYHEDRON Dimension:2
      Constraints:4  Equations:0  Rays:4  Lines:0
Constraints 4 4
Inequality: [  0  1 -2 ]
Inequality: [  1  0  0 ]
Inequality: [ -1  1  0 ]
Inequality: [  0  0  1 ]
Rays 4 4
Vertex: [  0  2 ]/1
Ray: [  0  1 ]
Ray: [  1  1 ]
Vertex: [  2  2 ]/1

```

testDomainConvex (JUnit test):

```

Validity Domain(0):
_dimension: 2
_nbConstraints: 4
_nbRays: 4
_nbEq: 0
_nbBid: 0

Inequality: 0 1 -2
Inequality: 1 0 0
Inequality: -1 1 0
Inequality: 0 0 1

Vertex: 0 2 /1
Ray: 0 1
Ray: 1 1
Vertex: 2 2 /1

```

testDomainCopy (C front-end test program):

```

POLYHEDRON Dimension:2
      Constraints:4  Equations:0  Rays:4  Lines:0
Constraints 4 4
Inequality: [  1  0  0 ]
Inequality: [ -1  1 -1 ]
Inequality: [  0  1 -2 ]
Inequality: [  0  0  1 ]
Rays 4 4
Ray: [  0  1 ]
Ray: [  1  1 ]
Vertex: [  1  2 ]/1
Vertex: [  0  2 ]/1
UNION POLYHEDRON Dimension:2
      Constraints:4  Equations:0  Rays:4  Lines:0
Constraints 4 4
Inequality: [  1  0 -1 ]
Inequality: [ -1  1  0 ]
Inequality: [  0  1 -2 ]
Inequality: [  0  0  1 ]
Rays 4 4
Ray: [  0  1 ]
Ray: [  1  1 ]
Vertex: [  2  2 ]/1
Vertex: [  1  2 ]/1

```

testDomainCopy (JUnit test):

```

Validity Domain(0):
_dimension: 2
_nbConstraints: 4
_nbRays: 4
_nbEq: 0
_nbBid: 0

Inequality: 1 0 0
Inequality: -1 1 -1
Inequality: 0 1 -2
Inequality: 0 0 1

Ray: 0 1
Ray: 1 1
Vertex: 1 2 /1
Vertex: 0 2 /1
Validity Domain(1):
_dimension: 2
_nbConstraints: 4
_nbRays: 4
_nbEq: 0
_nbBid: 0

Inequality: 1 0 -1
Inequality: -1 1 0
Inequality: 0 1 -2
Inequality: 0 0 1

Ray: 0 1
Ray: 1 1
Vertex: 2 2 /1
Vertex: 1 2 /1

```

testDomainDifference (C front-end test program):

```

POLYHEDRON Dimension:2
Constraints:3 Equations:1 Rays:2 Lines:0
Constraints 3 4
Equality: [ 1 0 0 ]
Inequality: [ 0 1 -2 ]
Inequality: [ 0 0 1 ]
Rays 2 4
Ray: [ 0 1 ]
Vertex: [ 0 2 ]/1

```

testDomainDifference (JUnit test):

```

Validity Domain(0):
_dimension: 2
_nbConstraints: 3
_nbRays: 2
_nbEq: 1
_nbBid: 0

Equality: 1 0 0
Inequality: 0 1 -2
Inequality: 0 0 1

Ray: 0 1
Vertex: 0 2 /1

```

testDomainImage (C front-end test program):

```

POLYHEDRON Dimension:2
      Constraints:4  Equations:0  Rays:4  Lines:0
Constraints 4 4
Inequality: [  0  1  0  ]
Inequality: [  0 -1  1  ]
Inequality: [ -1 -1  1  ]
Inequality: [  1  1  0  ]
Rays 4 4
Vertex: [  0  0  ]/1
Vertex: [ -1  1  ]/1
Vertex: [  0  1  ]/1
Vertex: [  1  0  ]/1

```

testDomainImage (JUnit test):

```

Validity Domain(0):
  _dimension:      2
  _nbConstraints:  4
  _nbRays:         4
  _nbEq:           0
  _nbBid:          0

Inequality:      0      1      0
Inequality:      0     -1      1
Inequality:     -1     -1      1
Inequality:      1      1      0

Vertex:          0      0      /1
Vertex:         -1      1      /1
Vertex:          0      1      /1
Vertex:          1      0      /1

```

testDomainIntersection (C front-end test program):

```

POLYHEDRON Dimension:2
      Constraints:3  Equations:0  Rays:3  Lines:0
Constraints 3 4
Inequality: [ -1  1 -1  ]
Inequality: [  1  0 -1  ]
Inequality: [  0  0  1  ]
Rays 3 4
Ray: [  1  1  ]
Ray: [  0  1  ]
Vertex: [  1  2  ]/1

```

testDomainIntersection (JUnit test):

```

Validity Domain(0):
  _dimension:      2
  _nbConstraints:  3
  _nbRays:         3
  _nbEq:           0
  _nbBid:          0

Inequality:     -1      1     -1
Inequality:      1      0     -1
Inequality:      0      0      1

Ray:            1      1
Ray:            0      1
Vertex:         1      2      /1

```

testDomainPreimage (C front-end test program):

```

POLYHEDRON Dimension:2
          Constraints:4  Equations:0  Rays:4  Lines:0
Constraints 4 4
Inequality: [ -1  0  1 ]
Inequality: [  2 -1  0 ]
Inequality: [  1 -2  2 ]
Inequality: [  0  1 -1 ]
Rays 4 4
Vertex: [  2  3 ]/2
Vertex: [  2  4 ]/3
Vertex: [  1  2 ]/2
Vertex: [  1  1 ]/1

```

testDomainPreimage (JUnit test):

```

Validity Domain(0):
  _dimension: 2
  _nbConstraints: 4
  _nbRays: 4
  _nbEq: 0
  _nbBid: 0

Inequality: -1  0  1
Inequality:  2 -1  0
Inequality:  1 -2  2
Inequality:  0  1 -1

Vertex: 2  3  /2
Vertex: 2  4  /3
Vertex: 1  2  /2
Vertex: 1  1  /1

```

testDomainSimplify (C front-end test program):

```

POLYHEDRON Dimension:3
          Constraints:3  Equations:2  Rays:2  Lines:1
Constraints 3 5
Equality: [  0  1  0  1 ]
Equality: [  0  0  1 -3 ]
Inequality: [  0  0  0  1 ]
Rays 2 5
Line: [  1  0  0 ]
Vertex: [  0 -1  3 ]/1

```

testDomainSimplify (JUnit test):

```

Validity Domain(0):
  _dimension: 3
  _nbConstraints: 3
  _nbRays: 2
  _nbEq: 2
  _nbBid: 1

Equality: 0  1  0  1
Equality: 0  0  1 -3
Inequality: 0  0  0  1

Line: 1  0  0
Vertex: 0 -1  3  /1

```

testDomainUnion (C front-end test program):

```

POLYHEDRON Dimension:2
  Constraints:4  Equations:0  Rays:4  Lines:0
Constraints 4 4
Inequality: [  1  0  0  ]
Inequality: [ -1  1 -1  ]
Inequality: [  0  1 -2  ]
Inequality: [  0  0  1  ]
Rays 4 4
Ray: [  0  1  ]
Ray: [  1  1  ]
Vertex: [  1  2  ]/1
Vertex: [  0  2  ]/1
UNION POLYHEDRON Dimension:2
  Constraints:4  Equations:0  Rays:4  Lines:0
Constraints 4 4
Inequality: [  1  0 -1  ]
Inequality: [ -1  1  0  ]
Inequality: [  0  1 -2  ]
Inequality: [  0  0  1  ]
Rays 4 4
Ray: [  0  1  ]
Ray: [  1  1  ]
Vertex: [  2  2  ]/1
Vertex: [  1  2  ]/1

```

testDomainUnion (JUnit test):

```

Validity Domain(0):
  _dimension: 2
  _nbConstraints: 4
  _nbRays: 4
  _nbEq: 0
  _nbBid: 0

Inequality: 1 0 0
Inequality: -1 1 -1
Inequality: 0 1 -2
Inequality: 0 0 1

Ray: 0 1
Ray: 1 1
Vertex: 1 2 /1
Vertex: 0 2 /1
Validity Domain(1):
  _dimension: 2
  _nbConstraints: 4
  _nbRays: 4
  _nbEq: 0
  _nbBid: 0

Inequality: 1 0 -1
Inequality: -1 1 0
Inequality: 0 1 -2
Inequality: 0 0 1

Ray: 0 1
Ray: 1 1
Vertex: 2 2 /1
Vertex: 1 2 /1

```

testEmptyPolyhedron (C front-end test program):

```

POLYHEDRON Dimension:2
              Constraints:3  Equations:3  Rays:0  Lines:0
Constraints 3 4
Equality:    [  1  0  0  ]
Equality:    [  0  1  0  ]
Equality:    [  0  0  1  ]
Rays 0 4

```

testEmptyPolyhedron (JUnit test):

```

Validity Domain(0):
  _dimension:    2
  _nbConstraints: 3
  _nbRays:       0
  _nbEq:         3
  _nbBid:        0

Equality:       1      0      0
Equality:       0      1      0
Equality:       0      0      1

```

testPolyhedron2Constraints (C front-end test program):

```

4 5
  0  1  0  0  -1
  0  0  1  0  1
  0  0  0  1  -3
  1  0  0  0  1

```

testPolyhedron2Constraint (JUnit test):

```

0 1 0 0 -1
0 0 1 0 1
0 0 0 1 -3
1 0 0 0 1

```

testPolyhedron2ParamDomain (C front-end test program):

```

nbV: 8
-----ParamDomain(0)-----
POLYHEDRON Dimension:3
              Constraints:4  Equations:0  Rays:4  Lines:0
Constraints 4 5
Inequality: [  0  1  0  0  ]
Inequality: [  1  0  0  0  ]
Inequality: [ -1  0  1  0  ]
Inequality: [  0  0  0  1  ]
Rays 4 5
Ray:       [  0  1  0  ]
Ray:       [  0  0  1  ]
Ray:       [  1  0  1  ]
Vertex:    [  0  0  0  ]/1
-----
-----ParamVertices(0)-----
<Vertex: 1 0 0 0 1
2 1 0 0 1
0 0 1 0 1
  | Domain: empty >
-----
-----ParamVertices(1)-----
<Vertex: 1 0 0 0 1

```

```

2 1 0 0 1
0 0 0 0 1
| Domain: empty >
-----
-----ParamVertices(2)-----
<Vertex: 1 0 0 0 1
0 0 0 0 1
0 0 1 0 1
| Domain: empty >
-----
-----ParamVertices(3)-----
<Vertex: 1 0 0 0 1
0 0 0 0 1
0 0 0 0 1
| Domain: empty >
-----
-----ParamVertices(4)-----
<Vertex: 0 0 0 0 1
0 1 0 0 1
-1 0 1 0 1
| Domain: empty >
-----
-----ParamVertices(5)-----
<Vertex: 0 0 0 0 1
0 1 0 0 1
0 0 0 0 1
| Domain: empty >
-----
-----ParamVertices(6)-----
<Vertex: 0 0 0 0 1
0 0 0 0 1
-1 0 1 0 1
| Domain: empty >
-----
-----ParamVertices(7)-----
<Vertex: 0 0 0 0 1
0 0 0 0 1
0 0 0 0 1
| Domain: empty >
-----

```

testPolyhedron2ParamDomain (JUnit test):

```

nbV: 8
-----ParamDomain(0)-----
Validity Domain(0):
_dimension: 3
_nbConstraints: 4
_nbRays: 4
_nbEq: 0
_nbBid: 0

Inequality: 0 1 0 0
Inequality: 1 0 0 0
Inequality: -1 0 1 0
Inequality: 0 0 0 1

Ray: 0 1 0
Ray: 0 0 1
Ray: 1 0 1
Vertex: 0 0 0 /1
-----
-----ParamVertices(0)-----
<Vertex: 1 0 0 0 1
2 1 0 0 1
0 0 1 0 1
| Domain: empty >
-----

```

```
-----ParamVertices(1)-----
<Vertex: 1 0 0 0 1
2 1 0 0 1
0 0 0 0 1
| Domain: empty >
-----
```

```
-----ParamVertices(2)-----
<Vertex: 1 0 0 0 1
0 0 0 0 1
0 0 1 0 1
| Domain: empty >
-----
```

```
-----ParamVertices(3)-----
<Vertex: 1 0 0 0 1
0 0 0 0 1
0 0 0 0 1
| Domain: empty >
-----
```

```
-----ParamVertices(4)-----
<Vertex: 0 0 0 0 1
0 1 0 0 1
-1 0 1 0 1
| Domain: empty >
-----
```

```
-----ParamVertices(5)-----
<Vertex: 0 0 0 0 1
0 1 0 0 1
0 0 0 0 1
| Domain: empty >
-----
```

```
-----ParamVertices(6)-----
<Vertex: 0 0 0 0 1
0 0 0 0 1
-1 0 1 0 1
| Domain: empty >
-----
```

```
-----ParamVertices(7)-----
<Vertex: 0 0 0 0 1
0 0 0 0 1
0 0 0 0 1
| Domain: empty >
-----
```

testPolyhedron2ParamVertices (C front-end test program):

```
-----Start testPolyhedron2Vertices-----
-----Start ParamPolyhedron-----
nbV: 8
-----ParamVertices(0)-----
<Vertex: 1 0 0 0 1
2 1 0 0 1
0 0 1 0 1
| Domain: 1 0 1 0 0
1 1 0 0 0
1 -1 0 1 0
1 0 0 0 1
-----
-----ParamVertices(1)-----
<Vertex: 1 0 0 0 1
2 1 0 0 1
0 0 0 0 1
| Domain: 1 0 1 0 0
```

```
1 1 0 0 0
1 -1 0 1 0
1 0 0 0 1
-----
-----ParamVertices (2)-----
<Vertex: 1 0 0 0 1
0 0 0 0 1
0 0 1 0 1
| Domain: 1 0 1 0 0
1 1 0 0 0
1 -1 0 1 0
1 0 0 0 1
-----
-----ParamVertices (3)-----
<Vertex: 1 0 0 0 1
0 0 0 0 1
0 0 0 0 1
| Domain: 1 0 1 0 0
1 1 0 0 0
1 -1 0 1 0
1 0 0 0 1
-----
-----ParamVertices (4)-----
<Vertex: 0 0 0 0 1
0 1 0 0 1
-1 0 1 0 1
| Domain: 1 0 1 0 0
1 -1 0 1 0
1 1 0 0 0
1 0 0 0 1
-----
-----ParamVertices (5)-----
<Vertex: 0 0 0 0 1
0 1 0 0 1
0 0 0 0 1
| Domain: 1 0 1 0 0
1 -1 0 1 0
1 1 0 0 0
1 0 0 0 1
-----
-----ParamVertices (6)-----
<Vertex: 0 0 0 0 1
0 0 0 0 1
-1 0 1 0 1
| Domain: 1 0 1 0 0
1 -1 0 1 0
1 1 0 0 0
1 0 0 0 1
-----
-----ParamVertices (7)-----
<Vertex: 0 0 0 0 1
0 0 0 0 1
0 0 0 0 1
| Domain: 1 0 1 0 0
1 -1 0 1 0
1 1 0 0 0
1 0 0 0 1
-----
-----End ParamPolyhedron-----
```

testPolyhedron2ParamVertices (JUnit test):

```
-----Start ParamPolyhedron-----  
nbV: 8
```

```
-----ParamVertices(0)-----  
<Vertex: 1 0 0 0 1  
2 1 0 0 1  
0 0 1 0 1  
| Domain: empty >  
-----
```

```
-----ParamVertices(1)-----  
<Vertex: 1 0 0 0 1  
2 1 0 0 1  
0 0 0 0 1  
| Domain: 1 0 1 0 0  
1 1 0 0 0  
1 -1 0 1 0  
1 0 0 0 1  
>  
-----
```

```
-----ParamVertices(2)-----  
<Vertex: 1 0 0 0 1  
0 0 0 0 1  
0 0 1 0 1  
| Domain: 1 0 1 0 0  
1 1 0 0 0  
1 -1 0 1 0  
1 0 0 0 1  
>  
-----
```

```
-----ParamVertices(3)-----  
<Vertex: 1 0 0 0 1  
0 0 0 0 1  
0 0 0 0 1  
| Domain: 1 0 1 0 0  
1 1 0 0 0  
1 -1 0 1 0  
1 0 0 0 1  
>  
-----
```

```
-----ParamVertices(4)-----  
<Vertex: 0 0 0 0 1  
0 1 0 0 1  
-1 0 1 0 1  
| Domain: 1 0 1 0 0  
1 -1 0 1 0  
1 1 0 0 0  
1 0 0 0 1  
>  
-----
```

```
-----ParamVertices(5)-----  
<Vertex: 0 0 0 0 1  
0 1 0 0 1  
0 0 0 0 1  
| Domain: 1 0 1 0 0  
1 -1 0 1 0  
1 1 0 0 0  
1 0 0 0 1  
>  
-----
```

```
-----ParamVertices(6)-----  
<Vertex: 0 0 0 0 1  
0 0 0 0 1  
>
```

```

-1 0 1 0 1
| Domain: 1 0 1 0 0
1 -1 0 1 0
1 1 0 0 0
1 0 0 0 1
>
-----
-----ParamVertices(7)-----
<Vertex: 0 0 0 0 1
0 0 0 0 1
0 0 0 0 1
| Domain: 1 0 1 0 0
1 -1 0 1 0
1 1 0 0 0
1 0 0 0 1
>
-----
-----End ParamPolyhedron-----

```

testPolyhedronEnumerate1 (C front-end test program):

```

-----Enumeration(0)-----
POLYHEDRON Dimension:1
          Constraints:2 Equations:0 Rays:2 Lines:0
Constraints 2 3
Inequality: [ -1  5 ]
Inequality: [  1 -3 ]
Rays 2 3
Vertex: [  3 ]/1
Vertex: [  5 ]/1

<--EValue
-- Enode size=4 pos=1 type=polynomial --
<--EValue fraction: 4/1 -->
<--EValue fraction: -19/3 -->
<--EValue fraction: 5/2 -->
<--EValue fraction: -1/6 -->
-- Enode --
-->
-----Enumeration-----

-----Enumeration(1)-----
POLYHEDRON Dimension:1
          Constraints:2 Equations:0 Rays:2 Lines:0
Constraints 2 3
Inequality: [  1 -5 ]
Inequality: [  0  1 ]
Rays 2 3
Ray: [  1 ]
Vertex: [  5 ]/1

<--EValue
-- Enode size=2 pos=1 type=polynomial --
<--EValue fraction: -16/1 -->
<--EValue fraction: 6/1 -->
-- Enode --
-->
-----Enumeration-----

```

testPolyhedronEnumerate1 (JUnit test):

```

-----Enumeration(0)-----
Validity Domain(0):
_dimension: 1
_nbConstraints: 2

```

```

_nbRays:      2
_nbEq:        0
_nbBid:       0

Inequality:   -1    5
Inequality:    1   -3

Vertex:       3    /1
Vertex:       5    /1

<--EValue
-- Enode size=4 pos=1 type=polynomial
  <--EValue fraction: 4/1 -->
  <--EValue fraction: -19/3 -->
  <--EValue fraction: 5/2 -->
  <--EValue fraction: -1/6 -->
-- Enode --
-->
-----Enumeration-----

-----Enumeration(1)-----
Validity Domain(0):
_dimension:   1
_nbConstraints: 2
_nbRays:      2
_nbEq:        0
_nbBid:       0

Inequality:   1    -5
Inequality:   0     1

Ray:          1
Vertex:       5    /1

<--EValue
-- Enode size=2 pos=1 type=polynomial
  <--EValue fraction: -16/1 -->
  <--EValue fraction: 6/1 -->
-- Enode --
-->
-----Enumeration-----

```

testPolyhedronEnumerate2 (C front-end test program):

```

-----Enumeration(0)-----

POLYHEDRON Dimension:1
                Constraints:2 Equations:0 Rays:2 Lines:0
Constraints 2 3
Inequality: [  1  -1  ]
Inequality: [  0   1  ]
Rays 2 3
Ray: [  1  ]
Vertex: [  1  ]/1

<--EValue
-- Enode size=3 pos=1 type=polynomial --
  <--EValue
  -- Enode size=2 pos=1 type=periodic --
    <--EValue fraction: 0/1 -->
    <--EValue fraction: -1/4 -->
  -- Enode --
  -->
  <--EValue fraction: 1/2 -->
  <--EValue fraction: 3/4 -->
  -- Enode --
-->
-----Enumeration-----

```

testPolyhedronEnumerate2 (JUnit test):

```

-----Enumeration(0)-----
Validity Domain(0):
_dimension:      1
_nbConstraints:  2
_nbRays:         2
_nbEq:           0
_nbBid:          0

Inequality:      1      -1
Inequality:      0       1

Ray:             1
Vertex:          1      /1

<--EValue
-- Enode size=3 pos=1 type=polynomial
<--EValue
-- Enode size=2 pos=1 type=periodic
  <--EValue fraction: 0/1 -->
  <--EValue fraction: -1/4 -->
-- Enode --
-->
<--EValue fraction: 1/2 -->
<--EValue fraction: 3/4 -->
-- Enode --
-->
-----Enumeration-----

```

testPolyhedronScan (C front-end test program):

```

POLYHEDRON Dimension:4
      Constraints:3  Equations:0  Rays:5  Lines:2
Constraints 3 6
Inequality: [ -1  0  1  0  0 ]
Inequality: [  1  0  0  0 -1 ]
Inequality: [  0  0  0  0  1 ]
Rays 5 6
Line: [  0  1  0  0 ]
Line: [  0  0  0  1 ]
Ray: [  1  0  1  0 ]
Ray: [  0  0  1  0 ]
Vertex: [  1  0  1  0 ]/1
UNION POLYHEDRON Dimension:4
      Constraints:3  Equations:0  Rays:5  Lines:2
Constraints 3 6
Inequality: [  0 -1  1  0  0 ]
Inequality: [  0  1  0  0 -1 ]
Inequality: [  0  0  0  0  1 ]
Rays 5 6
Line: [  1  0  0  0 ]
Line: [  0  0  0  1 ]
Ray: [  0  1  1  0 ]
Ray: [  0  0  1  0 ]
Vertex: [  0  1  1  0 ]/1

```

testPolyhedronScan (JUnit test):

```

Validity Domain(0):
_dimension:      4
_nbConstraints:  3
_nbRays:         5
_nbEq:           0
_nbBid:          2

Inequality:      -1      0      1      0      0

```

```

Inequality:  1    0    0    0    -1
Inequality:  0    0    0    0     1

Line:       0    1    0    0
Line:       0    0    0    1
Ray:        1    0    1    0
Ray:        0    0    1    0
Vertex:     1    0    1    0    /1
Validity Domain(1):
_dimension:  4
_nbConstraints: 3
_nbRays:     5
_nbEq:       0
_nbBid:      2

Inequality:  0   -1    1    0    0
Inequality:  0    1    0    0   -1
Inequality:  0    0    0    0    1

Line:       1    0    0    0
Line:       0    0    0    1
Ray:        0    1    1    0
Ray:        0    0    1    0
Vertex:     0    1    1    0    /1

```

testRays2Polyhedron (C front-end test program):

```

POLYHEDRON Dimension:3
          Constraints:4  Equations:0  Rays:4  Lines:0
Constraints 4 5
Inequality: [  0    1    0   -1 ]
Inequality: [  1   -1    0    0 ]
Inequality: [ -1    0    1    0 ]
Inequality: [  0    0    0    1 ]
Rays 4 5
Vertex: [  1    1    1 ]/1
Ray:    [  0    0    1 ]
Ray:    [  1    0    1 ]
Ray:    [  1    1    1 ]

```

testRays2Polyhedron (JUnit test):

```

Validity Domain(0):
_dimension:  3
_nbConstraints: 4
_nbRays:     4
_nbEq:       0
_nbBid:      0

Inequality:  0    1    0   -1
Inequality:  1   -1    0    0
Inequality: -1    0    1    0
Inequality:  0    0    0    1

Vertex:     1    1    1    /1
Ray:        0    0    1
Ray:        1    0    1
Ray:        1    1    1

```

testUniversePolyhedron (C front-end test program):

```

POLYHEDRON Dimension:3
          Constraints:1 Equations:0 Rays:4 Lines:3
Constraints 1 5
Inequality: [ 0 0 0 1 ]
Rays 4 5
Line: [ 1 0 0 ]
Line: [ 0 1 0 ]
Line: [ 0 0 1 ]
Vertex: [ 0 0 0 ]/1

```

testUniversePolyhedron (JUnit test):

```

Validity Domain(0):
_dimension: 3
_nbConstraints: 1
_nbRays: 4
_nbEq: 0
_nbBid: 3

Inequality: 0 0 0 1

Line: 1 0 0
Line: 0 1 0
Line: 0 0 1
Vertex: 0 0 0 /1

```

testAffineHermite (C front-end test program):

```

3 3
 6 0 3
 0 2 1
 0 0 1

```

testAffineHermite (JUnit test):

```

6 0 3
0 2 1
0 0 1

```

testGetHermite (C front-end test program):

```

5 5
 1 0 0 0 0
 0 4 0 0 0
 0 0 3 0 0
 0 0 0 1 0
 0 3 6 3 10

```

testGetHermite (JUnit test):

```

1 0 0 0 0
0 4 0 0 0
0 0 3 0 0
0 0 0 1 0
0 3 6 3 10

```

testGetSmithNormalProduct (C front-end test program):

```

4 4
 1  0  0  0
 0  1  0  0
 0  0  2  0
 0  0  0  2

```

testGetSmithNormalProduct (JUnit test):

```

1 0 0 0
0 1 0 0
0 0 2 0
0 0 0 2

```

testGetSmithNormalU (C front-end test program):

```

6 6
 1  0  0  0  0  0
 0  1  0  0  0  0
 0  0  0  0  0  1
 0  0 -1  0  0  0
 0  0 -1  0  1  0
 0  0  0 -1  0  0

```

testGetSmithNormalU (JUnit test):

```

1 0 0 0 0 0
0 1 0 0 0 0
0 0 0 0 0 1
0 0 -1 0 0 0
0 0 -1 0 1 0
0 0 0 -1 0 0

```

testGetSmithNormalV (C front-end test program):

```

8 8
 1  0  0  0  0  0 -2  0  6
 0  1 -2  0  0  0 -6  0  0
 0  0  1  0  0  0  3  0  0
 0  0  0  1  0  1  0 -3  0
 0  0  0  0  0  0  1  0  0
 0  0  0  0  0  0 -1  0  3
 0  0  0  0  1  0  0  0  0
 0  0  0  0  0  0  0  0  1

```

testGetSmithNormalV (JUnit test):

```

1 0 0 0 0 -2 0 6
0 1 -2 0 0 0 -6 0
0 0 1 0 0 0 3 0
0 0 0 1 0 1 0 -3
0 0 0 0 0 0 1 0
0 0 0 0 0 -1 0 3
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1

```

testGetUniModular (C front-end test program):

```
4 4
 1  0  1 -1
 1 -1  0  6
 0  0 -1  4
 0  1  0 -4
```

testGetUniModular (JUnit test):

```
1 0 1 -1
1 -1 0 6
0 0 -1 4
0 1 0 -4
```

3 Literature List

Piplib-1.3.6 manual http://www.prism.uvsq.fr/~cedb/bastools/pip_manual/index.html

Polylib-5.22.0 manual <http://www.irisa.fr/polylib/DOC/index.html>

Expat-2.0.0 manual <http://expat.sourceforge.net/>

O'Reilly Expat introduction <http://www.xml.com/pub/a/1999/09/expat/index2.html>

O'Reilly – XML CD bookshelf 2nd Edition

4 Word list

API: Abbreviation of *application program interface*, a set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer puts the blocks together.

Expat : **Expat** is a stream-oriented XML 1.0 parser library, written in C. Expat was one of the first open source XML parsers and has been incorporated into many open source projects, including the Apache HTTP Server, Mozilla, Perl, Python and PHP.

JNI: (Java Native Interface) A Java programming interface, or API, that allows developers to access the languages of a host system and determine the way Java integrates with native code. The JNI has been a point of contention between Sun and Microsoft, since Microsoft seeks to create its own native code interface and Sun claims this violates their licensing agreement.

JVM : Acronym for *Java Virtual Machine*. An abstract computing machine, or virtual machine, JVM is a platform-independent execution environment that converts Java bytecode into machine language and executes it. Most programming languages compile source code directly into machine code that is designed to run on a specific microprocessor architecture or operating system, such as Windows or UNIX. A JVM -- a machine within a machine -- mimics a real Java processor, enabling Java bytecode to be executed as actions or operating system calls on any processor regardless of the operating system. For example, establishing a socket connection from a workstation to a remote machine involves an operating system call. Since different operating systems handle sockets in different ways, the JVM translates the programming code so that the two machines that may be on different platforms are able to connect.

SAX : Short for *Simple API for XML*, an event-based API that, as an alternative to DOM, allows someone to access the contents of an XML document. SAX was originally a Java-only API. The current version supports several programming language environments other than Java.

SAX was developed by the members of the XML-DEV mailing list.

Piplib: PIP/PipLib is the well known Paul Feautrier's parametric integer linear programming solver. PIP is software which finds the lexicographic minimum of the set of integer points lying inside a convex polyhedron. This polyhedron can depend linearly on one or more integral parameters. If the user asks for a non integral solution, PIP can give the exact solution as an integral quotient. The heart of PIP is the parametrized Gomory's cuts algorithm followed by the parameterized dual simplex method. The PIP Library (PipLib for short) was implemented to allow the user to call PIP directly from his

programs, without file accesses or system calls. The user only needs to link his programs with C libraries.

Polylib: The Polyhedral Library (PolyLib for short) operates on objects made up of unions of polyhedra of any dimension. It was first developed by Doran Wilde at IRISA, in Rennes, France, in connection with the ALPHA project. This first version (1.1) manipulates non parameterized unions of polyhedra through the following operations: intersection, difference, union, convex hull, simplify, image and preimage, plus some input and output functions. The polyhedra are computed in their dual implicit and Minkowski representations, in homogeneous spaces.

Version 2 of the PolyLib included parameterized vertices computation. PolyLib3.14 includes Ehrhart polynomials computation, which permits to count the number of integer points contained in a parameterized polyhedron. PolyLib4 included the GNU MP library (as a compilation option), and 64 bits computations, in order to avoid integer overflows. Polylib5 is a merge of Strasbourg, Rennes and BYU Polylib

XML : Short for *Extensible Markup Language*, a specification developed by the W3C. XML is a pared-down version of SGML, designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations.

5 UML Drawings

The following is a listing of UML drawings for some of the Java classes involved in this project:

<p>PolylibInterface</p> <ul style="list-style-type: none"> + Constraints2Polyhedron (SignedMatrix): Polyhedron + ConstraintsSimplify (SignedMatrix domain, SignedMatrix context): SignedMatrix + DomainConvex (Polyhedron): Polyhedron + DomainCopy (Polyhedron): Polyhedron + DomainDifference (Polyhedron, Polyhedron): Polyhedron + DomainPreimage (Polyhedron, JMatrix): Polyhedron + DomainUnion (Polyhedron, Polyhedron): Polyhedron + EmptyPolyhedron (int): Polyhedron + Polyhedron2Constraints (Polyhedron): SignedMatrix + Polyhedron2ParamDomain (Polyhedron, Polyhedron): ParamPolyhedron + Polyhedron2ParamVertices (Polyhedron, Polyhedron): ParamPolyhedron + PolyhedronEnumerate (Polyhedron, Polyhedron): Enumeration + LexSmallerEnumerate (SignedMatrix, SignedMatrix, int, SignedMatrix): Enumeration + PolyhedronScan (Polyhedron, Polyhedron): Polyhedron + Rays2Polyhedron (SignedMatrix): Polyhedron + UniversePolyhedron (int): Polyhedron + DomainImage (Polyhedron, JMatrix): Polyhedron + DomainIntersection (Polyhedron, Polyhedron): Polyhedron + PolyhedronCount (Polyhedron, EvaluateFunction): void + DomainSimplify (Polyhedron, Polyhedron): Polyhedron 	<p>Polylib</p> <ul style="list-style-type: none"> + Constraints2Polyhedron (SignedMatrix): Polyhedron + ConstraintsSimplify (SignedMatrix domain, SignedMatrix context): SignedMatrix + DomainConvex (Polyhedron): Polyhedron + DomainCopy (Polyhedron): Polyhedron + DomainDifference (Polyhedron, Polyhedron): Polyhedron + DomainPreimage (Polyhedron, JMatrix): Polyhedron + DomainUnion (Polyhedron, Polyhedron): Polyhedron + EmptyPolyhedron (int): Polyhedron + Polyhedron2Constraints (Polyhedron): SignedMatrix + Polyhedron2ParamDomain (Polyhedron, Polyhedron): ParamPolyhedron + Polyhedron2ParamVertices (Polyhedron, Polyhedron): ParamPolyhedron + PolyhedronEnumerate (Polyhedron, Polyhedron): Enumeration + LexSmallerEnumerate (SignedMatrix, SignedMatrix, int, SignedMatrix): Enumeration + PolyhedronScan (Polyhedron, Polyhedron): Polyhedron + Rays2Polyhedron (SignedMatrix): Polyhedron + UniversePolyhedron (int): Polyhedron + DomainImage (Polyhedron, JMatrix): Polyhedron + DomainIntersection (Polyhedron, Polyhedron): Polyhedron + PolyhedronCount (Polyhedron, EvaluateFunction): void + DomainSimplify (Polyhedron, Polyhedron): Polyhedron <p>- _polylib: PolylibInterface + _instance: PolyLib</p>
<p>ZPolylibInterface</p> <ul style="list-style-type: none"> + LatticeDifference(Lattice, Lattice): LatticeUnion + LatticeImage(Lattice, JMatrix): Lattice + LatticeIncludes(Lattice, Lattice): boolean + LatticeIntersection(Lattice, Lattice): Lattice + LatticePreimage(Lattice, JMatrix): Lattice + LatticeSimplify(LatticeUnion): LatticeUnion + ZDomainCopy(ZPolyhedron): ZPolyhedron + ZDomainDifference(ZPolyhedron, ZPolyhedron): ZPolyhedron + ZDomainImage(ZPolyhedron, JMatrix): ZPolyhedron + ZDomainIncludes(ZPolyhedron, ZPolyhedron): boolean + ZDomainIntersection(ZPolyhedron, ZPolyhedron): ZPolyhedron + ZDomainPreimage(ZPolyhedron, JMatrix): ZPolyhedron + ZDomainSimplify(ZPolyhedron): ZPolyhedron + ZDomainUnion(ZPolyhedron, ZPolyhedron): ZPolyhedron + affineHermite(Lattice): Lattice + getHermite(SignedMatrix): SignedMatrix + getSmithNormalProduct(SignedMatrix): SignedMatrix + getSmithNormalU(SignedMatrix): SignedMatrix + getSmithNormalV(SignedMatrix): SignedMatrix + getUniModular(SignedMatrix): SignedMatrix + isEmptyLattice(Lattice): boolean + isEmptyZPolyhedron(ZPolyhedron): boolean 	<p>ZPolyLib</p> <ul style="list-style-type: none"> + getInstance(): getInstance + LatticeDifference(Lattice, Lattice): LatticeUnion + LatticeImage(Lattice, JMatrix): Lattice + LatticeIncludes(Lattice, Lattice): boolean + LatticeIntersection(Lattice, Lattice): Lattice + LatticePreimage(Lattice, JMatrix): Lattice + LatticeSimplify(LatticeUnion): LatticeSimplify + ZDomainCopy(ZPolyhedron): ZPolyhedron + ZDomainDifference(ZPolyhedron, ZPolyhedron): ZPolyhedron + ZDomainImage(ZPolyhedron, JMatrix): ZPolyhedron + ZDomainIncludes(ZPolyhedron, ZPolyhedron): boolean + ZDomainIntersection(ZPolyhedron, ZPolyhedron): ZPolyhedron + ZDomainPreimage(ZPolyhedron, JMatrix): ZPolyhedron + ZDomainSimplify(ZPolyhedron): ZPolyhedron + ZDomainUnion(ZPolyhedron, ZPolyhedron): ZPolyhedron + affineHermite(Lattice): Lattice + getHermite(SignedMatrix): SignedMatrix + getSmithNormalProduct(SignedMatrix): SignedMatrix + getSmithNormalU(SignedMatrix): SignedMatrix + getSmithNormalV(SignedMatrix): SignedMatrix + getUniModular(SignedMatrix): SignedMatrix + isEmptyLattice(Lattice): boolean + isEmptyZPolyhedron(ZPolyhedron): boolean + latticeIntersection(Lattice, Lattice): Lattice + zdomainDifference(ZPolyhedron, ZPolyhedron): ZPolyhedron + zdomainIntersection(ZPolyhedron, ZPolyhedron): ZPolyhedron + zdomainUnion(ZPolyhedron, ZPolyhedron): ZPolyhedron - ZPolyLib() <p>- _zpolylib: ZPolylibInterface - _instance: ZPolyLib</p>

ZPolyhedron

```

+ ZPolyhedron()
+ ZPolyhedron(Lattice, Polyhedron)
+ ZPolyhedron(ZPolyhedron)
+ add(ZPolyhedron): void
+ getLattice(): Lattice
+ getPolyhedron(): Polyhedron
+ hasNext(): boolean
+ next(): ZPolyhedron
+ setLattice(Lattice): void
+ setPolyhedron(Polyhedron): void
+ toString(): String

```

```

+ Lat: Lattice
+ Poly: Polyhedron
+ head: ZPolyhedron
+ next: Zpolyhedron
+ tail: ZPolyhedron

```

ParamVertices

```

+ ParamVertices()
+ add(ParamVertices): void
+ setParameters(Vector): void
+ hasNext(): boolean
+ next():ParamVertices
+ setNext(ParamVertices): void
+ setDomain(SignedMatrix): void
+ getDomain():SignedMatrix
+ setVertex(SignedMatrix): void
+ getVertex():SignedMatrix
+ setHead(ParamVertices): void
+ getHead():ParamVertices
+ setTail(ParamVertices): void
+ getTail():ParamVertices
+ getCoordinates(): Vector
+ toString(): String
+ dump(): String
+ dumpAll(): void
+ vertices()

```

```

- _domain : SignedMatrix
- _head : ParamVertices
- _next : ParamVertices
- _tail : ParamVertices
- _vertex : SignedMatrix
- _parameters : Vector

```

MappingFunction

```

MappingFunction(SignedMatrix)
+ evaluate(): void
+ setValue(int, int): void
+ toString():String

```

```

- _array : Vector
- _result : Vector
- _matrix : SignedMatrix

```

PolylibException

```

+ PolylibException(String)
+ getMessage(): String

```

```

- _message : String

```

ParamPolyhedron

```

+ ParamPolyhedron()
+ setParamDomain(ParamDomain): void
+ getParamDomain():ParamDomain
+ setParamVertices(ParamVertices): void
+ getParamVertices():ParamVertices
+ setNbV(int): void
+ dump(): void

```

```

- _D : ParamDomain
- _V : ParamVertices
- _nbV : int

```

ParamDomain

```

+ ParamDomain()
+ getDomain():ParamDomain
+ setDomain(Polyhedron): void
+ initF(int): void
+ getF(): boolean[]
+ setF(boolean[]): void
+ addToF(int, boolean): void
+ getNext():ParamDomain
+ setNext(ParamDomain): void
+ dump(): void

```

```

- _F : boolean[]
- _domain : Polyhedron
- _next : ParamDomain

```

EvaluateFunction

```

+ evaluate(): void
+ setValue(int, int): void

```

LatticeUnion

```

LatticeUnion()
LatticeUnion(Lattice)
LatticeUnion(LatticeUnion)
+ LatUnion(): Iterator
+ add(LatticeUnion): void
+ getLattice(): Lattice
+ hasNext(): boolean
+ next():LatticeUnion
+ setLattice(Lattice): void
+ toString():String

```

```

+ Lat : Lattice
+ head : LatticeUnion
+ next : LatticeUnion
+ tail : LatticeUnion

```

ENode

```
ENode(int, int, String)
+ clear(): void
+ getArr(): EValue[]
+ getPos(): int
+ getSize(): int
+ getType(): String
+ isPseudo(): boolean
+ setEValue(int, EValue): void
+ toOneLineString(Vector): String
+ toPolynomial(Vector, String): String
+ toString(Vector): String
```

```
- _arr : EValue[]
- _pos : int
- _size : int
- _type : String
```

Lattice

```
Lattice()
Lattice(int, int)
Lattice(String, int, int)
Lattice(SignedMatrix)
Lattice(String)
```

EValue

```
EValue()
EValue(ENode)
EValue(int, int)
+ clear(): void
+ getD(): int
+ setD(int): void
+ getN(): int
+ setN(int): void
+ getP(): ENode
+ setP(ENode): void
+ isPseudo(): boolean
+ toOneLineString(Vector): String
+ toPolynomial(Vector, String): String
+ toString(Vector): String
```

```
- _d : int
- _n : int
- _p : ENode
```

Enumeration

```
Enumeration()
Enumeration(Polyhedron, EValue)
+ add(Enumeration): void
+ clear(): void
+ domains(): Iterator
+ eToString(): String
+ isZero(): boolean
+ toPolynomial(Vector): String
+ toString(Vector): String
+ toString(): String
+ getEV(): EValue
+ setEV(EValue): void
+ getHead(): Enumeration
+ setHead(Enumeration): void
+ getNext(): Enumeration
+ setNext(Enumeration): void
+ getTail(): Enumeration
+ setTail(Enumeration): void
+ getValidityDomain(): Polyhedron
+ setValidityDomain(Polyhedron): void
```

```
- EV : EValue
- _ValidityDomain : Polyhedron
- _head : Enumeration
- _next : Enumeration
- _tail : Enumeration
- _paramVector : Vector
```

countFunction

```
CountFunction()
+ evaluate(): void
+ setValue(int, int): void
+ getVector(): Vector
```

```
- _array : Vector
- _total : Vector
```

Polyhedron

```
Polyhedron()
Polyhedron(Polyhedron)
Polyhedron(int, int, int, int, int)
+ add(Polyhedron): void
+ clear(): void
+ domains(): Iterator
+ enumerateToString(): String
+ getConstraint(int, int): long
+ getRay(int, int): long
+ hasNext(): boolean
+ next(): Polyhedron
+ initConstraint(int, int): void
+ initRay(int, int): void
+ setConstraint(int, int, long): void
+ setRay(int, int, long): void
+ toString(): String
+ getDimension(): int
+ setDimension(int): void
+ getHead(): Polyhedron
+ setHead(Polyhedron): void
+ getNbBid(): int
+ setNbBid(int): void
+ getNbConstraints(): int
+ setNbConstraints(int): void
+ getNbEq(): int
+ setNbEq(int): void
+ getNbRays(): int
+ setNbRays(int): void
+ getNext(): Polyhedron
+ setNext(Polyhedron): void
+ getRay(): long[][]
+ setRay(long[][]): void
+ getTail(): Polyhedron
+ setTail(Polyhedron): void
```

```
- _dimension : int
- _nbBid : int
- _nbConstraints : int
- _nbEq : int
- _nbRays : int
- _constraint : long[][]
- _head : Polyhedron
- _next : Polyhedron
- _ray : long[][]
- _tail : Polyhedron
```