LEMA Definition And Implementation

Philippe Théveny

ÉPI ARÉNAIRE

20 May 2010

Plan

1. Introduction
2. The Library
3. The Language

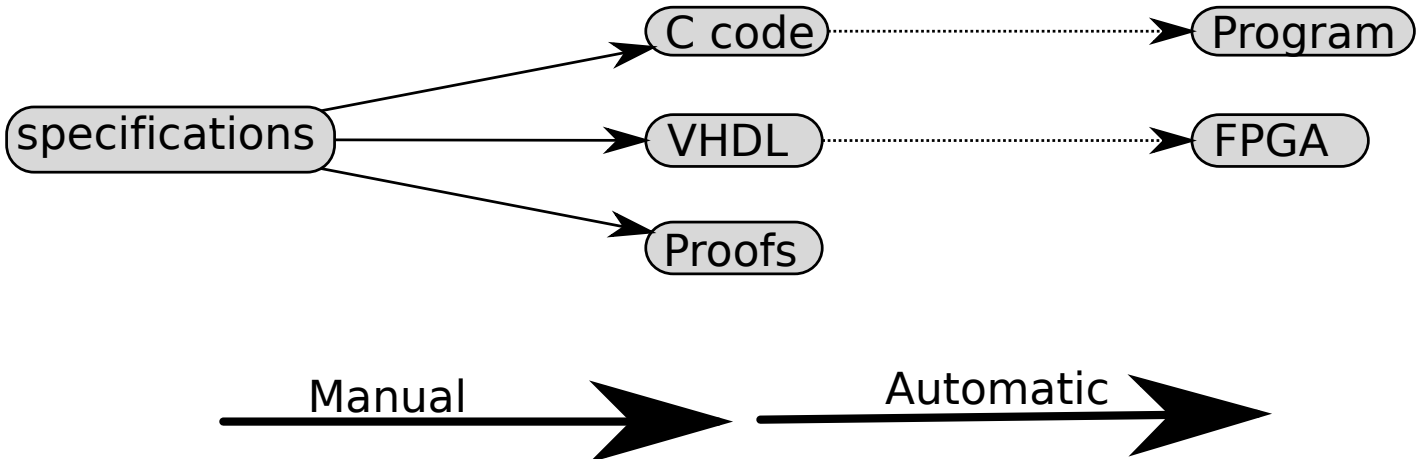
1 Introduction
- Context
- ARÉNAIRE's Tools For Numerical Analysis
- CAS For Numerical Analysis
- Tools Integration

2 The Library

3 The Language

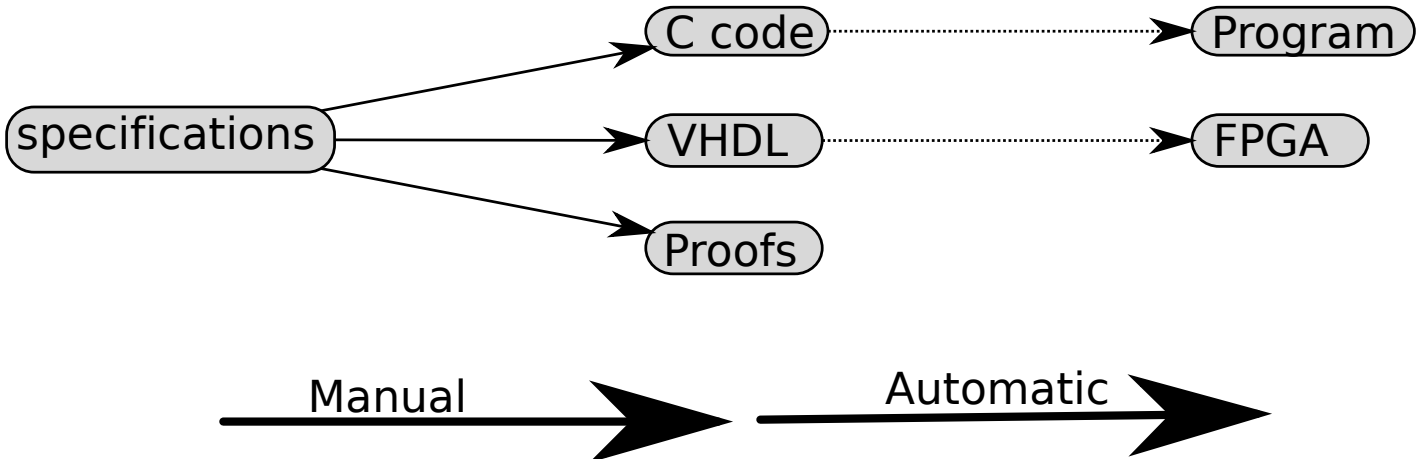
Context

Implementing efficient and certified numerical programs is partly handmade work, partly an automatic process.



Context

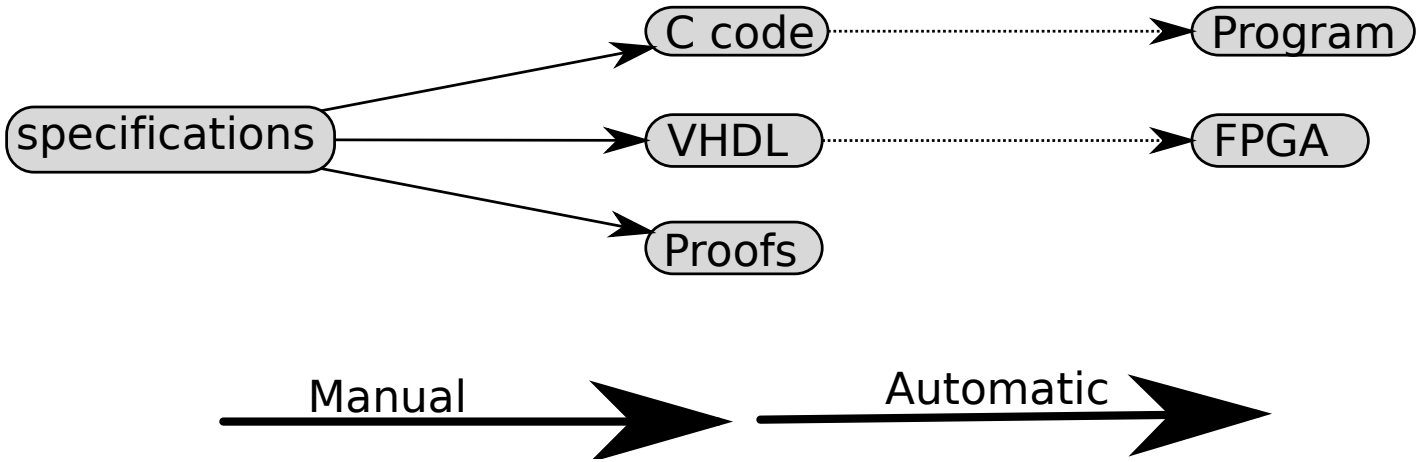
Implementing efficient and certified numerical programs is partly handmade work, partly an automatic process.



We intend to automate the whole process.

Context

Implementing efficient and certified numerical programs is partly handmade work, partly an automatic process.



We intend to automate the whole process.
Some existing tools may help.

ARÉNAIRE's Tools For Numerical Analysis

Specific useful tools were developped by the ARÉNAIRE team

- Sollya
  - Certify a bound on the infinite norm of an elementary function
  - Compute various polynomial approximations
  - Some symbolic manipulations of expressions

ARÉNAIRE's Tools For Numerical Analysis

Specific useful tools were developped by the ARÉNAIRE team

- Sollya
  - Certify a bound on the infinite norm of an elementary function
  - Compute various polynomial approximations
  - Some symbolic manipulations of expressions
- Gappa
  - Formal proofs of arithmetic properties.

ARÉNAIRE's Tools For Numerical Analysis

Specific useful tools were developped by the ARÉNAIRE team

- Sollya
  - Certify a bound on the infinite norm of an elementary function
  - Compute various polynomial approximations
  - Some symbolic manipulations of expressions
- Gappa
  - Formal proofs of arithmetic properties.
- CGPE
  - Generation of fast and certified codes for the evaluation of bivariate polynomials.

CAS For Numerical Analysis

Mathematical properties of the function are useful

- Argument reduction uses symmetry and functional relations.
- Bound on range values may be easily computed from the derivative.
- Algorithm convergence may require monotonicity.

CAS For Numerical Analysis

Mathematical properties of the function are useful

- Argument reduction uses symmetry and functional relations.
- Bound on range values may be easily computed from the derivative.
- Algorithm convergence may require monotonicity.

This can be done with the help of a Computer Algebra System.

Tools Integration

Tools characteristics

- CAS, Sollya, CGPE, and Gappa are command-line tools.
- They do not share a common language.

Tools Integration

Tools characteristics

- CAS, Sollya, CGPE, and Gappa are command-line tools.
- They do not share a common language.

How to automate the process?

1 Introduction

2 The Library
- Overall Architecture
- Input
- Communication
- Output
- Current State

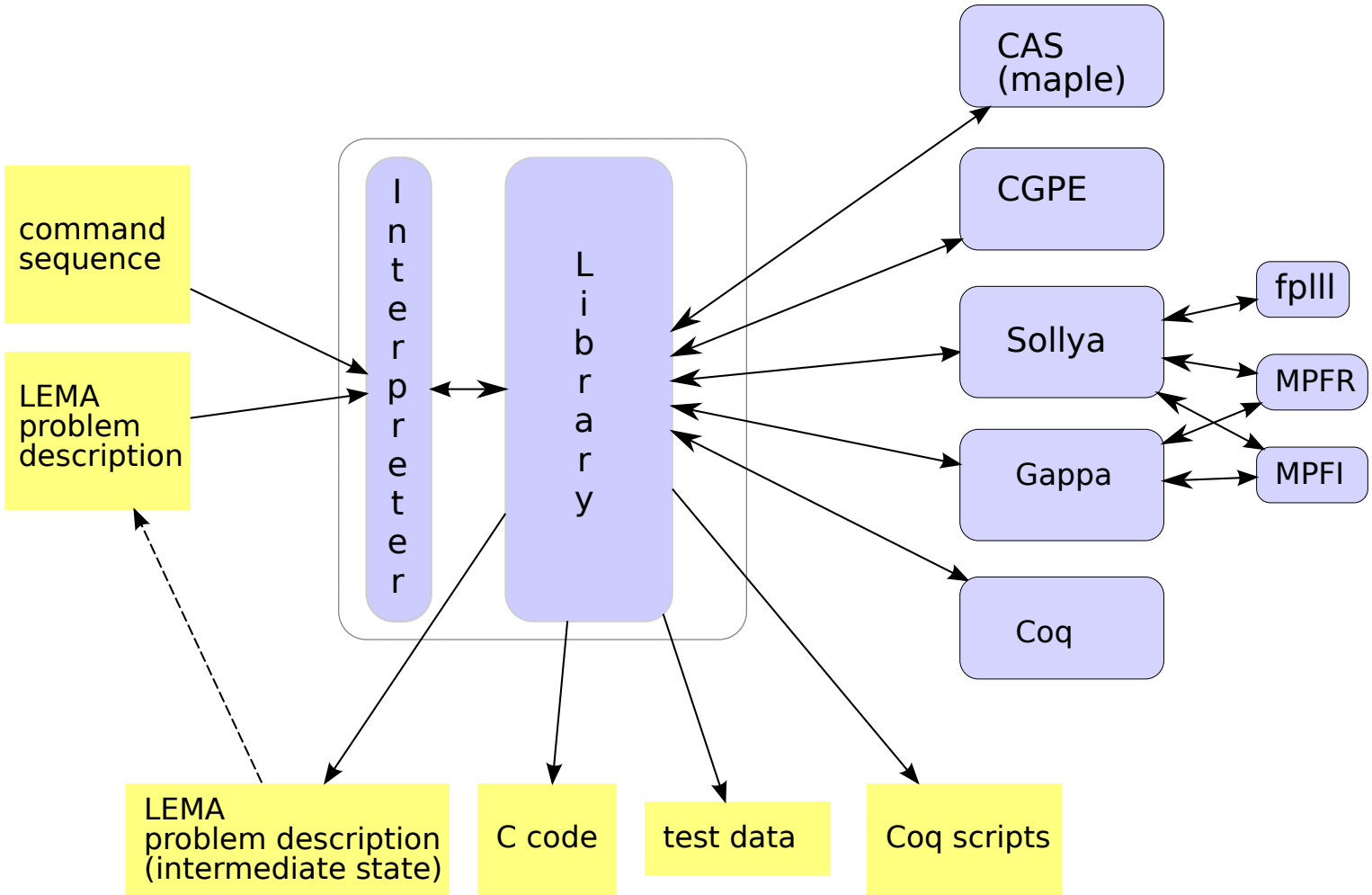
3 The Language

Overall Architecture

We are designing

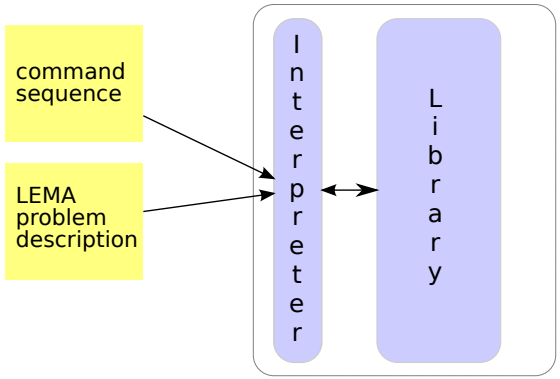
- a new language LEMA (*Langage pour les Expressions Mathématiques Annotées*), so as to integrate in one place all information produced by external tools
- a new tool modifying documents written in LEMA in interaction with external tools

Overall Architecture



The tool itself consists in an interpreter and a library.

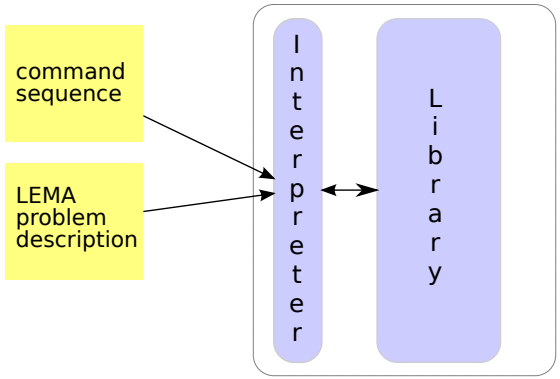
Input



An implementation problem is split in two parts

- specifications written in LEMA
- command sequence

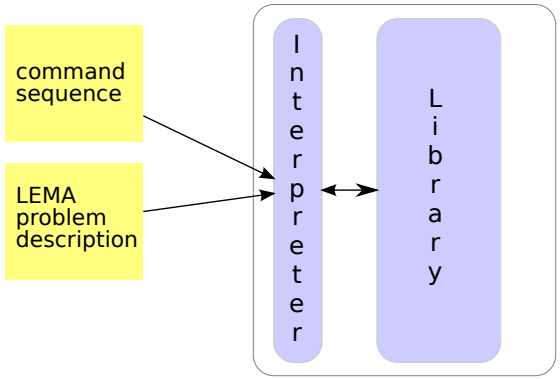
Input



An implementation problem is split in two parts

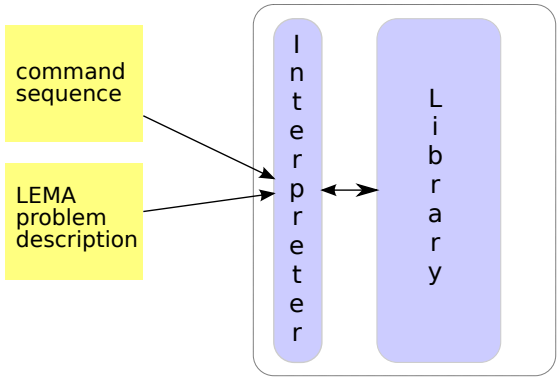
- specifications = the problem
- command sequence = its solution

Input



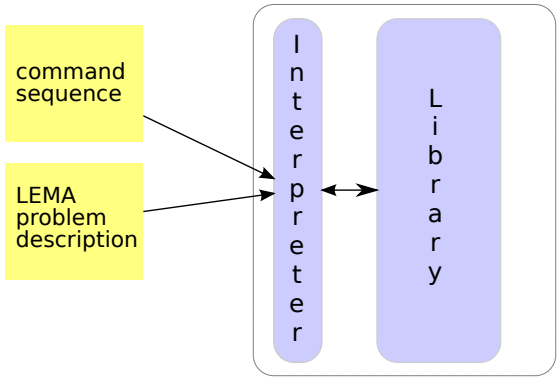
The library reads the problem description file and represents specification data in an internal tree.

Input



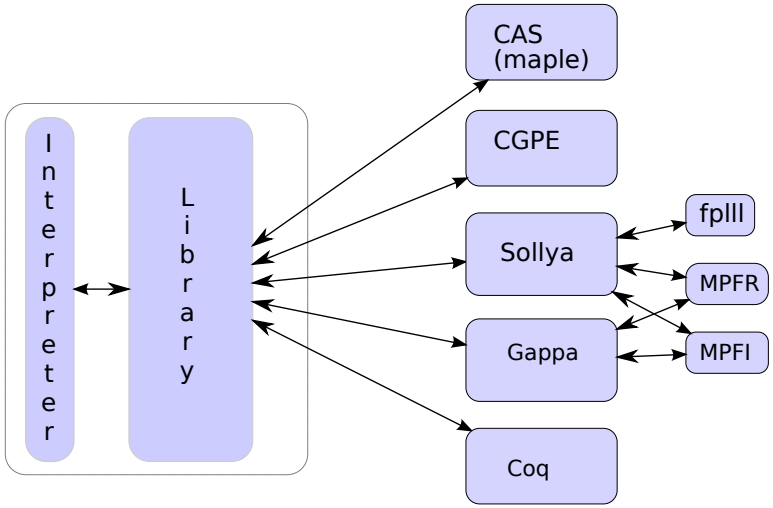
The interpreter reads the commands in turn and call the corresponding library functions who act on the internal representation.

Input



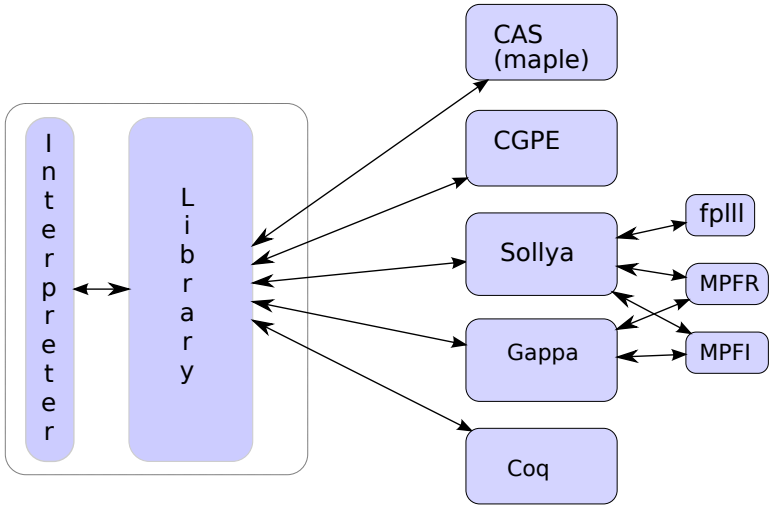
Library functions are also available to programs through its application public interface.

Communication



Communications are done in text mode.

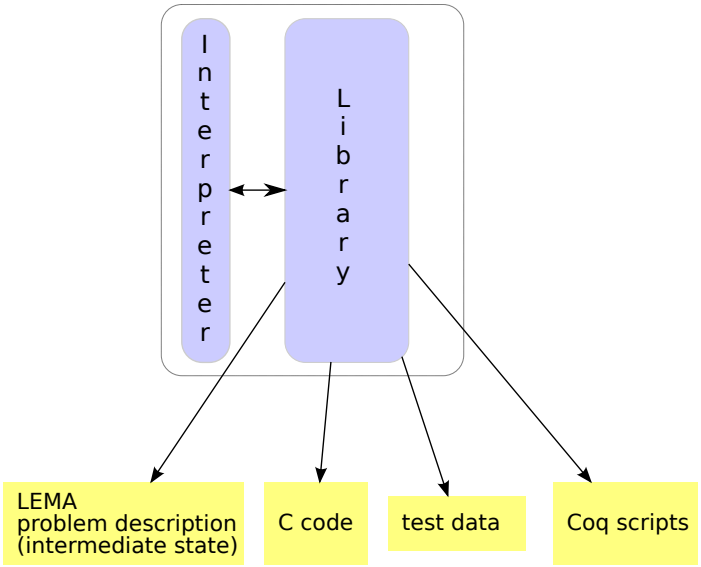
Communication



When sending a request to an external tool, the library

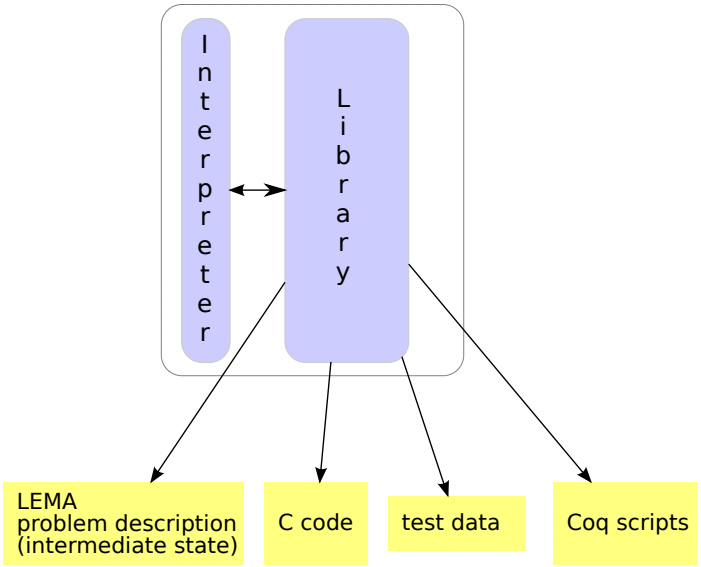
1. selects relevant data,
2. formats them in a form that the tool understand,
3. requests computation on them,
4. add answer to the data set.

Output



Data in intermediate state should be representable in LEMA.

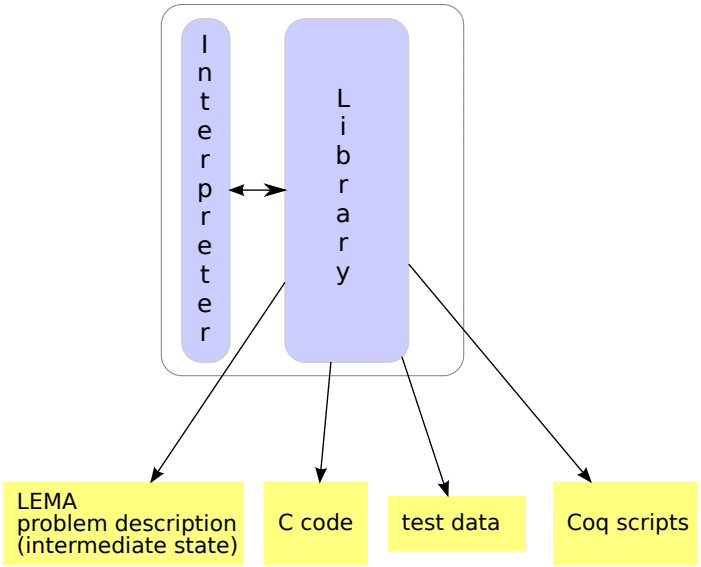
Output



The main goals

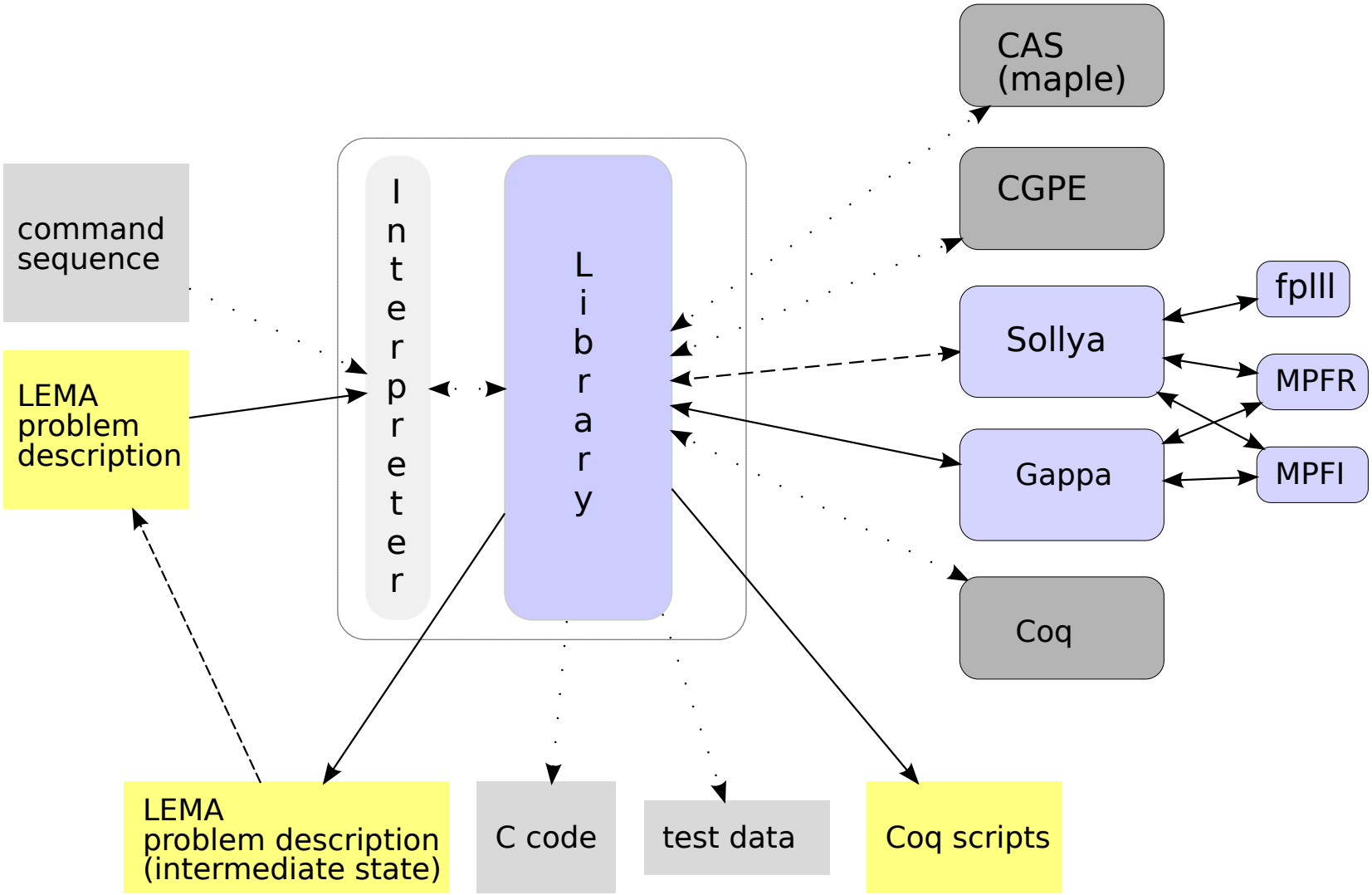
- generating C code
- generating proofs as Gappa scripts, Coq scripts or HOL scripts

Output



From the specification it will be easy to generate data tests.

Current State Of Work



The interface with Sollya is work in progress.

1 Introduction

2 The Library

3 The Language

- Why a new language?
- Main Idea
- MathML
- LEMA

Why a new language?

We want a language with sufficient expressiveness to state

- specifications
- data computed with external tools

Expressing Specifications

The specifications are about

- the function to be implemented
  - its mathematical expression
  - its expected output on special values
- the types of input, output, and intermediate variables
- arithmetics associated with these types
- target platform capacities
  - instruction set
  - parallelism
- hints for proof assistants

Integrating Derived Data

We want to link new data to initial ones, that is in particular

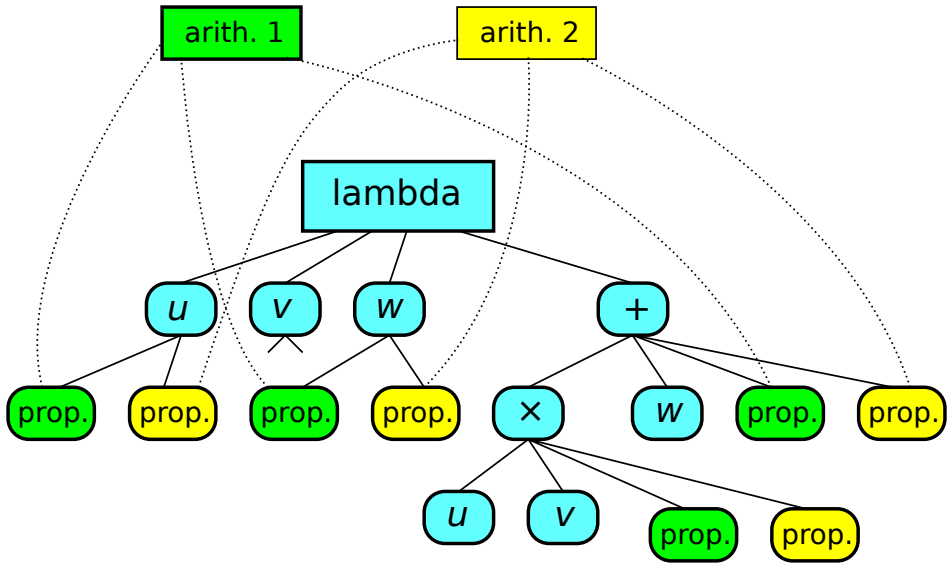
- to bind mathematically equivalent expressions
- to bind a polynomial approximation to the original expression
- to bind evaluation in a particular arithmetic to the expression being evaluated
- to store properties of such evaluations
- to record proof of properties

Main Idea

We chose to develop a MathML-based language.

Main Idea

We chose to develop a MathML-based language.
Nodes are annotated with properties depending on the arithmetic used for calculation.



MathML

MathML design goals (as expressed in the language specification)

- Encode both mathematical notation and mathematical meaning.
- Facilitate conversion to and from other mathematical formats, both presentational and semantic.
- Allow the passing of information intended for specific renderers and applications.
- Provide for extensibility.
- Be legible to humans, and simple for software to generate and process.

MathML

MathML design goals (as expressed in the language specification)

- Encode both mathematical notation and mathematical meaning.
- Facilitate conversion to and from other mathematical formats, both presentational and semantic.
- Allow the passing of information intended for specific renderers and applications.
- Provide for extensibility.
- Be legible to humans, and simple for software to generate and process.

Example In MathML

The interval $[0.17, 1071481116960651033753473963881151744232652 8]$ encoded in MathML

Example

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
 <interval>
  <cn id="left" type="real">0.17</cn>
  <cn id= "right" type="integer">
   1071481116960651033753473963881151744232652 8
  </cn>
 </interval>
</math>
```

<cn> stands for *content number*.

The <semantics>, <annotation> Pair

MathML allows several encodings for the same element

Example

```
<semantics>
  <apply>
    <plus>
    <apply>
      <sin/>
      <ci>x</ci>
    </apply>
    <cn>5</cn>
  </apply>
  <annotation encoding="application/x-tex">
    \sin x + 5
  </annotation>
</semantics>
```

Benefits And Drawbacks Of Using MathML

LEMA reuses the content part of MathML because

- it is easy to write mathematical formulae,
- the tree structure of mathematical expression does not dictate an order of evaluation,
- some tools already understand MathML.

Benefits And Drawbacks Of Using MathML

LEMA reuses the content part of MathML because

- it is easy to write mathematical formulae,
- the tree structure of mathematical expression does not dictate an order of evaluation,
- some tools already understand MathML.

But MathML was not designed to encode all possible hardware numbers and floating-point properties.

Floating-point Numbers In LEMA

We define new elements in a custom namespace. The rounding to nearest in single precision of the right endpoint 1071481116960651033753473963881151744232652 8 is encoded as

Example

```
<cn id="right_Binary32_Nearest"
 lema:type="Binary32"
 lema:rounding="Nearest"
 lema:exact="true"
 lema:overflow="true">+0x7bp+136</cn>
```

Floating-point Numbers In LEMA

We define new elements in a custom namespace. The rounding to nearest in single precision of the right endpoint 1071481116960651033753473963881151744232652 8 is encoded as

Example

```
<cn id="right_Binary32_Nearest"
 lema:type="Binary32"
 lema:rounding="Nearest"
 lema:exact="true"
 lema:overflow="true">+0x7bp+136</cn>
```

Floating-point Numbers In LEMA

We define new elements in a custom namespace. The rounding to nearest in single precision of the right endpoint 1071481116960651033753473963881151744232652 8 is encoded as

Example

```
<cn id="right_Binary32_Nearest"
 lema:type="Binary32"
 lema:rounding="Nearest"
 lema:exact="true"
 lema:overflow="true">+0x7bp+136</cn>
```

Here, attributes realize the annotations with floating-point properties.

Rounding Numbers In LEMA

To link a number to its rounding, we use the <semantics>, <annotation-xml> elements

Example

```
<semantics>
 <cn id="left" type="real">0.17</cn>
 <annotation-xml lema:type="Binary32_Nearest"
  encoding="application/lema-evaluation+xml">
  <cn id="left_Binary32_Nearest"
   lema:type="Binary32"
   lema:rounding="Nearest"
   lema:exact="false">+0xae147bp-26</cn>
 </annotation-xml>
</semantics>
```

Rounding Numbers In LEMA

Several roundings may be attached to the initial number.
The `lema:type` attribute distinguishes them.

Example

```
<semantics>
 <cn>...</cn>
 <annotation-xml lema:type="Binary32_Nearest"
  encoding="application/lema-evaluation+xml">
  <cn>...</cn> </annotation-xml>
 <annotation-xml lema:type="Binary32_Zero"
  encoding="application/lema-evaluation+xml">
  <cn>...</cn> </annotation-xml>
 <annotation-xml lema:type="Binary64_Nearest"
  encoding="application/lema-evaluation+xml">
  <cn>...</cn> </annotation-xml>
</semantics>
```

Proofs In LEMA

Proofs can be stored in a `lema:proof` element directly in the document

Example

```
  <lema:proof href="right_Binary32_Nearest" type="gappa">
<![CDATA[
@rndn = float< 24, -126, ne >;
MaxFloat = 0xf.fffffp+124;
right = 107148111696065103375347396388115174423265 28;
right_Binary32_Nearest = +0x7bp+136;

{
 right_Binary32_Nearest - rndn(right) in [0, 0]
 /\ right_Binary32_Nearest - right in [0, 0]
 /\ right_Binary32_Nearest - MaxFloat >= 0
}
]]>
  </lema:proof>
```

Proofs In LEMA

They refer to the number and its rounding through their `id` attribute

Example

```
  <lema:proof href="right_Binary32_Nearest" type="gappa">
<![CDATA[
@rndn = float< 24, -126, ne >;
MaxFloat = 0xf.fffffp+124;
right = 10714811169606510337534739638811517442326528;
right_Binary32_Nearest = +0x7bp+136;

{
 right_Binary32_Nearest - rndn(right) in [0, 0]
 /\ right_Binary32_Nearest - right in [0, 0]
 /\ right_Binary32_Nearest - MaxFloat >= 0
}
]]>
  </lema:proof>
```

Proofs In LEMA

Proofs can be saved in external files to preserve them from accidental changes

Example

```
<lema:proof href="left_Binary32_Nearest"
 type="gappa"
 src="left_Binary32_Nearest.gappa"/>
<lema:proof href="left_Binary32_Nearest"
 type="coq"
 src="left_Binary32_Nearest.v"/>
```

Proofs In LEMA

Proofs of floating-point properties are linked to the number rounding through a reference to its `id` attribute

Example

```
<semantics>
 <cn id="left">0.17</cn>

 <annotation-xml lema:type="Binary32_Nearest"
  encoding="application/lema-evaluation+xml">

  <cn id="left_Binary32_Nearest"
   lema:exact="false"> +0xae147bp-26 </cn>
  <lema:proof href="left_Binary32_Nearest"
   type="gappa"  src="left_Binary32_Nearest.gappa"/>

 </annotation-xml>
</semantics>
```

Questions?