LEMA: Towards a Language for Reliable Arithmetic

Philippe Théveny.
Joint work with Vincent Lefèvre, Florent de Dinechin, Claude-Pierre Jeannerod, Christophe Mouilleron, David Pfannholzer, Nathalie Revol

PLMMS 2010, 8th July
Plan

1. Context
   - Motivations
   - A Typical Problem
   - Software Architecture

2. Description of LEMA

3. A Simple Example

4. Conclusion
Motivations for a new language: LEMA

LEMA stands for “Langage pour les Expressions Mathématiques Annotées”
Motivations for a new language: LEMA

LEMA stands for “Langage pour les Expressions Mathématiques Annotées”

Our long term goal is
“To generate automatically certified and efficient numerical code.”
LEMA stands for “Langage pour les Expressions Mathématiques Annotées”

Our long term goal is “To generate automatically certified and efficient numerical code.”

- numerical code: typically, a floating-point function written in C implementing a mathematical operation.
Motivations for a new language: LEMA

LEMA stands for “Langage pour les Expressions Mathématiques Annotées”

Our long term goal is “To generate automatically certified and efficient numerical code.”

- numerical code: typically, a floating-point function written in C implementing a mathematical operation.
- efficient code: the characteristics of the target system are taken into account so as to improve performance.
LEMA stands for “Langage pour les Expressions Mathématiques Annotées”

Our long term goal is “To generate automatically certified and efficient numerical code.”

- numerical code: typically, a floating-point function written in C implementing a mathematical operation.
- efficient code: the characteristics of the target system are taken into account so as to improve performance.
- certified code: mathematical properties used during the implementation process have to be proved.
LEMA stands for “Langage pour les Expressions Mathématiques Annotées”

Our long term goal is
“To generate automatically certified and efficient numerical code.”

- numerical code: typically, a floating-point function written in C implementing a mathematical operation.
- efficient code: the characteristics of the target system are taken into account so as to improve performance.
- certified code: mathematical properties used during the implementation process have to be proved.
Implementation Sketch

A typical problem of implementation

- mathematical scalar function
- special values

choose a "good" floating-point polynomial approximation

choose a "good" evaluation scheme

translate in C
Implementation Sketch

A typical problem of implementation

mathematical scalar function $\rightarrow$ choose a "good" floating-point polynomial approximation

maximum degree acceptable error bound $\rightarrow$ choose a "good" evaluation scheme $\rightarrow$ translate in C
A typical problem of implementation

mathematical scalar function $\rightarrow$ choose a "good" floating-point polynomial approximation

maximum degree acceptable error bound ...

Sollya

choose a "good" evaluation scheme $\rightarrow$ translate in C
Implementation Sketch

A typical problem of implementation

mathematical scalar function $\rightarrow$ choose a "good" floating-point polynomial approximation $\rightarrow$ choose a "good" evaluation scheme $\rightarrow$ translate in C

target parallelism
numerical quality
...

We want to automate the process of generating such C code with proofs.
A typical problem of implementation

- mathematical scalar function
- choose a "good" floating-point polynomial approximation
- choose a "good" evaluation scheme
- target parallelism
- numerical quality
- ...

CGPE

We want to automate the process of generating such C code with proofs.
Implementation Sketch

A typical problem of implementation

- mathematical scalar function
- choose a "good" floating-point polynomial approximation
- choose a "good" evaluation scheme
- target parallelism
- numerical quality
- ... 

GAPPA

We want to automate the process of generating such C code with proofs.
A typical problem of implementation

1. Choose a “good” floating-point polynomial approximation
2. Choose a “good” evaluation scheme
3. Translate in C

We want to automate the process of generating such C code with proofs.
Implementation Sketch

A typical problem of implementation

mathematical scalar function $\rightarrow$ choose a "good" floating-point polynomial approximation $\rightarrow$ choose a "good" evaluation scheme $\rightarrow$ translate in C

Sollya  CGPE  GAPPA

We want to automate the process of generating such C code with proofs.
Tool Integration

- CAS (Maple)
- CGPE
- Sollya
- Gappa
- Coq
Tool Integration

Library

- CAS (Maple)
- CGPE
- Sollya
- Gappa
- Coq
Tool Integration

- Problem description
- Library
- CAS (Maple)
- CGPE
- Sollya
- Gappa
- Coq
Tool Integration

- **Problem Description**
- **Library**
  - **CAS (Maple)**
  - **CGPE**
  - **Sollya**
  - **Gappa**
  - **Coq**
Tool Integration

Library

- Problem description
- Internal representation
- C code
- Proofs

Tools:
- CAS (Maple)
- CGPE
- Sollya
- Gappa
- Coq
Tool Integration

problem description

written in LEMA

C code

Proofs

Library

internal representation

CAS (Maple)

CGPE

Sollya

Gappa

Coq
Plan

1. Context

2. Description of LEMA
   - Requirements
   - Overview of a LEMA Document
   - MathML
   - Annotating Mathematical Expressions

3. A Simple Example

4. Conclusion
We want a language with sufficient expressiveness to state
- specifications
- data computed with external tools
Requirements

We want a language with sufficient expressiveness to state

- specifications
  - 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
- hints for proof assistants
- data computed with external tools
We want a language with sufficient expressiveness to state

- specifications
- data computed with external tools
  - to bind mathematically equivalent expressions
  - to bind a polynomial approximation to the original function
  - to store evaluation properties
  - to record proof of properties
We chose to develop LEMA as an XML-application.
Overview of a LEMA Document

We chose to develop LEMA as an XML-application.

Integrating derived data produced by external tools
We chose to develop LEMA as an XML-application.

Locality of information in a tree:
- useful particular data come from ancestors
Overview of a LEMA Document

We chose to develop LEMA as an XML-application.

Locality of information in a tree:
- useful particular data come from ancestors
- context data are found near the root node
Overview of a LEMA Document

We chose to develop LEMA as an XML-application.

We use content MathML for mathematical expressions.

- common communication language between mathematical tools
- extensible by design
Example in MathML

The interval \([0.17, 10714811169606510337534739638811517442326528]\) encoded in MathML

Example

```xml
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <interval>
    <cn id="left" type="real">0.17</cn>
    <cn id="right" type="integer">10714811169606510337534739638811517442326528</cn>
  </interval>
</math>

<cn> stands for content number.
```
The `<semantics>, <annotation> Pair

MathML allows several encodings for the same element

Example

```xml
<semantics>
  <apply>
    <plus/>
    <apply>
      <sin/>
      <ci>x</ci>
    </apply>
    <cn>5</cn>
  </apply>
  <annotation encoding="application/x-tex">
    \sin x + 5
  </annotation>
</semantics>
```
A closer look at a subtree
Polynomial Approximation

Semantics: binary relation
Polynomial Approximation

Function: f
Degree: d
Domain: D
"good" approximation
Norm: infnorm
Error bound: epsilon

The relation is not binary!
Polynomial Approximation

Even worse in a floating-point context
Polynomial Approximation

Let us choose a particular axis and use annotations for anything else
Plan

1. Context

2. Description of LEMA

3. A Simple Example

4. Conclusion
Rounding a Real Number to a Floating-point Number

Real Number: x

Floating-point Format: Binary32

Rounding Mode: ToNearest

Evaluate to

Floating-point Number: u

How to embed in the LEMA document

- the floating-point number $u$ representing $x$ in IEEE-754 Binary32 format, rounded to nearest
How to embed in the LEMA document

- the floating-point number $u$ representing $x$ in IEEE-754 Binary32 format, rounded to nearest
- additional rounding properties with their proofs
The rounding to nearest in single precision of 10714811169606510337534739638811517442326528 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.
We define new attributes in a custom namespace

Example

\[
\text{<cn id="right\_Binary32\_Nearest" lema:type="Binary32"}
\text{lema:rounding="Nearest"}
\text{lema:exact="true"}
\text{lema:overflow="true"}>+0x7bp+136</cn>
\]
Floating-point Numbers in LEMA

We define new attributes in a custom namespace

Example

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

**Example**

```xml
<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>
```
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" ...>
    <cn>...</cn>
  </annotation-xml>

  <annotation-xml lema:type="Binary32_Zero" ...>
    <cn>...</cn>
  </annotation-xml>

  <annotation-xml lema:type="Binary64_Nearest" ...>
    <cn>...</cn>
  </annotation-xml>
</semantics>
Proofs in LEMA

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

Example

```xml
<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

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

Example

```xml
<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 can be saved in external files to preserve them from accidental changes

Example

```xml
<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>
Plan

1. Context
2. Description of LEMA
3. A Simple Example
4. Conclusion
Current state

- This is work in progress!
- We are defining new vocabulary on the fly
- There is no formal grammar of LEMA (yet)
Current state

- This is work in progress!
- We are defining new vocabulary on the fly
- There is no formal grammar of LEMA (yet)

Further possible developments (from the language point of view)

- Formalize a grammar for validation
- Formalize floating-point specific concepts in an OpenMath content dictionary