An Efficient Computer Algebra System for Python

Pearu Peterson
pearu.petersen@gmail.com

Laboratory of Systems Biology, Institute of Cybernetics, Estonia
Simula Research Laboratory, Norway

- Introduction
- Design criteria
- Sympycore architecture
  - Implementation notes
  - Handling infinities
- Performance comparisons
- Conclusions

SIAM Conference on Computational Science and Engineering
March 2-6, 2009
Miami, Florida
1. Introduction

What is CAS?

Computer Algebra System (CAS) is a software program that facilitates symbolic mathematics.

The core functionality of a CAS is manipulation of mathematical expressions in symbolic form.

[Wikipedia]

Our aim — provide a package to manipulate mathematical expressions within a Python program.

Target applications — code generation for numerical applications, arbitrary precision computations, etc in Python.
Existing tools — Wikipedia lists more than 40 CAS-es:

- commercial/free,
- full-featured programs/problem specific libraries,
- in-house, C, C++, Haskell, Java, Lisp, etc programming languages.

from core functionality to a fully-featured system

Possible approaches

- wrap existing CAS libraries to Python:

- create interfaces to CAS programs:
  Pythonica (Mathematica, 2004), SAGE[GPL] (Maxima, Axiom, Maple, Mathematica, MuPAD, etc.)

- write a CAS package from scratch:
  Sympy[BSD], Sympycore[BSD], Pymbolic[?] (2008), PySymbolic[LGPL] (2000), etc
2. Design criteria

Symbolic expressions in silica . . .

• memory efficient representation
• memory and CPU efficient manipulation
• support variety of mathematical concepts — algebraic approach
• extensibility is crucial — from core to fully-featured system
• separate core implementation details from library algorithms
Symbolic expressions

- atomic expressions — symbols, numbers
- composite expressions — unevaluated operations
- multiple representations possible

Representation of symbolic expressions consists of . . .

- Data structures to store operands
- Methods/functions to interpret data structures
- Classes to define algebraic properties

For example, \( x \ast y \) can be represented as

\[
\text{Ring}(\text{MUL}, [x, y])
\]

or as

\[
\text{CommutativeRing}(\text{BASE_EXP_DICT}, \{x: 1, y: 1\})
\]
3. Sympycore architecture

- Symbolic expressions are instances of `Algebra` subclasses.
- An `Algebra` instance holds pair of `head` and `data` parts:
  \[
  \langle \text{Algebra} \rangle (\langle \text{head part} \rangle, \langle \text{data part} \rangle)
  \]
- The `head` part holds operation methods.
- The `data` part holds operands.
- The `Algebra` class defines valid operation methods like `__mul__`, `__add__`, etc. that apply the corresponding operation methods (in `head`) to operands (in `data`).
3.1. Atomic heads

SYMBOL — data is arbitrary object (usually a string), \( \langle Algebra \rangle \) instance represents any element of the corresponding algebraic structure:

\[
x = Algebra(SYMBOL, 'x')
\]

NUMBER — data is numeric object, \( \langle Algebra \rangle \) instance represents a concrete element of the corresponding algebraic structure:

\[
r = Algebra(NUMBER, 3.14)
\]
3.2. Arithmetic heads

**ADD** — data is a list of operands to unevaluated addition operation:
\[
\text{Ring}(\text{ADD}, [x, y]) \rightarrow x + y
\]

**MUL** — data is a list of operands to unevaluated multiplication operation:
\[
\text{Ring}(\text{MUL}, [x, y]) \rightarrow x \times y
\]

**POW** — data is a tuple of base and exponent:
\[
\text{Ring}(\text{POW}, (x, y)) \rightarrow x^{\ast\ast} y
\]

**TERM_COEFF** — data is a tuple of symbolic term and numeric coefficient:
\[
\text{Ring}(\text{TERM_COEFF}, (x, 2)) \rightarrow 2 \times x
\]

**TERM_COEFF_DICT** — data is a dictionary of term-coefficient pairs:
\[
\text{Ring}(\text{TERM_COEFF_DICT}, \{x: 2, y: 3\}) \rightarrow 2*x + 3*y
\]

**BASE_EXP_DICT** — data is a dictionary of base-exponent pairs:
\[
\text{CommutativeRing}(\text{BASE_EXP_DICT}, \{x: 2, y: 3\}) \rightarrow x^{\ast\ast}2 * y^{\ast\ast}3
\]

**EXP_COEFF_DICT** — data contains polynomial symbols and a dictionary of exponents-coefficient pairs:
\[
\text{Ring}(\text{EXP_COEFF_DICT}, \text{Pair}((x, y), \{(2,0): 3, (5,6): 7\})) \rightarrow 3*x^{\ast\ast}2 + 7*x^{\ast\ast}5*y^{\ast\ast}6
\]
3.3. Other heads

NEG, POS, SUB, DIV, MOD — verbatim arithmetic heads:
  \[ \text{Ring(SUB, [x, y, z])} \rightarrow x - y - z \]

INVERT, BOR, BXOR, BAND, LSHIFT, RSHIFT — binary heads

LT, LE, GT, GE, EQ, NE — relational heads:
  \[ \text{Logic(LT, (x, y))} \rightarrow x < y \]

NOT, AND, OR, XOR, EQUIV, IMPLIES, IS, IN — logic heads:
  \[ \text{Logic(OR, (x, y))} \rightarrow x \text{ or } y \]

APPLY, SUBSCRIPT, LAMBDA, ATTR, KWARG — functional heads:
  \[ \text{Ring(Apply, (f, (x, y)))} \rightarrow f(x, y) \]

SPECIAL, CALLABLE — special heads

MATRIX — sparse matrix heads

UNION, INTERSECTION, SETMINUS — set heads

TUPLE, LIST, DICT — container heads

...
3.4. Algebra classes

Expr
   Algebra
      Verbatim
      Ring
         CommutativeRing
            Calculus
            Unit
            FunctionRing
           MatrixRing
      Logic
      Set
     ...

3.5. Examples

> from sympycore import *
> x, y, z = map(Calculus, 'xyz')
> 3*x + y + x/2
Calculus('y + 7/2*x')
> (x+y)**2
Calculus('(y + x)**2')
> ((x+y)**2).expand()
Calculus('2*y*x + x**2 + y**2')
from sympycore.physics import meter
x*meter+2*meter
Unit('(x + 2)*m')

f = Function('f')
f+sin
FunctionRing_Calc_to_Calc('Sin + f')
(f+sin)(x)
Calculus('Sin(x) + f(x)')

m=Matrix([[1,2], [3,4]])
print m.inv() * m
1 0
0 1
print m.A * m
1 4
9 16

Logic('x>1 and a and x>1')
Logic('a and x>1')
4. Implementation notes

Circular imports — modules implement initialization functions that are called when all subpackages are imported to initialize any module objects

Immutability of composites containing mutable types —
 Expr instance .is_writable — True if hash is not computed yet.

\[
\begin{align*}
\text{hash}(&\langle \text{dict} \rangle) = \text{hash}(&\text{frozenset}(&\langle \text{dict} \rangle.\text{items}())) \\
\text{hash}(&\langle \text{list} \rangle) = \text{hash}(&\text{tuple}(&\langle \text{list} \rangle))
\end{align*}
\]

Equality tests Expr .as_lowlevel() — used in hash computations and in equality tests.

```python
>>> Calculus(TERM_COEFF_DICT, {}).as_lowlevel()
0
>>> Calculus(TERM_COEFF_DICT, {x:1}).as_lowlevel()
Calculus('x')
>>> Calculus(TERM_COEFF_DICT, {x:1, y:1}).as_lowlevel()
(TERM_COEFF_DICT, {Calculus('x'): 1, Calculus('y'): 1})
```
5. **Infinity problems**

In most computer algebra systems handling infinities is inconsistent:

\[ 2 \cdot x \cdot \infty \rightarrow x \cdot \infty \]

but

\[ x \cdot \infty + x \cdot \infty \rightarrow 2 \cdot x \cdot \infty \]

\[ \text{expand}((x + 2) \cdot \infty) \rightarrow \infty + x \cdot \infty \]

incorrect if \( x = -1 \).
5.1. Sympycore Infinity

Sympycore defines \texttt{Infinity} object to represent extended numbers such as directional infinities and undefined symbols in a consistent way.

\textbf{Definition:} \texttt{Infinity}(d) = \lim_{r \to \infty} (r \times d), d \in \mathbb{C}

\textbf{Operations with finite numbers:}

\texttt{Infinity}(d) \ < \ op \ > \ n = \lim_{r \to \infty} (r \times d \ < \ op \ > \ n)

\textbf{Operations with infinite numbers:}

\texttt{Infinity}(d_1) \ < \ op \ > \texttt{Infinity}(d_2) = \lim_{r_1 \to \infty, r_2 \to \infty} (r_1 \times d_1 \ < \ op \ > r_2 \times d_2)

>>> oo = \texttt{Infinity}(1)
>>> x*oo - x*oo
\texttt{Infinity(\texttt{Calculus(’EqualArg(x, -x)*x’))}}

always correctly evaluates to \texttt{undefined=Infinity}(0).

>>> x*oo + y
\texttt{Infinity(\texttt{Calculus(’x*(1 + (EqualArg(x, y) - 1)*IsUnbounded(y))’}})
Performance comparisons

Performance history of Python based CAS-s
Executing: 3*(1/2*x + 2/3*y + 4/5*z) -> 3/2*x + 2*y + 12/5*z

- SymPy SVN
- sympy-research branch
- sympy-sandbox branch
- Sympy Core SVN: 35 000 execs/sec
- GiNaC: 180 000 execs/sec
- Sage(libSingular): 110 000 execs/sec
- Maxima(GCL): 19 400 execs/sec
- pyginac: 15 700 execs/sec
- swiginac: 13 400 execs/sec
- Maxima(clisp): 2 320 execs/sec
- SymPy, devel: 1030 execs/sec
- symbolic: 540 execs/sec
- Sage(Maxima): 35 execs/sec
6. Conclusions

• Sympycore — a research project, its aim is to seek out new high-performance solutions to represent and manipulate symbolic expressions in Python language

• — fastest Python based CAS core implementation

• — uses algebraic approach, supporting various mathematical concepts is equally easy

http://sympycore.google.com
Pearu Peterson
Fredrik Johansson