

Re: Computer Algebra Algorithms

Source: <http://sci.tech-archive.net/Archive/sci.math.symbolic/2005-04/msg00030.html>

- *From:* [JohnCreighton @xxxxxxxxxxx](#)
 - *Date:* 8 Apr 2005 11:15:53 -0700
-

Bernard Parisse wrote:

>> I've been trying to follow this thread somewhat but there is many
>> more posts here then I want to toughly dissect. I don't really
think
>> in the above posts enough motivation was given behind what you want
to
>> do. If you want to learn CAS, learn lisp because that is what the
open
>> source CAS are programmed in and a large part of the academic work
in
>> AI uses lisp.
>>
>
> Actually, there are C++ programmed CAS, like my giac/xcas
> www-fourier.ujf-grenoble.fr/~parisse/giac.html
> They are more flexible for programmers because they are available
> as C++ libraries, not only as a binary.

You're link doesn't work. Do I get from what you are saying you
code is not open source? If it is not open source how are people going
to learn the algorithms by exploring the code? How does your system
compare to Maxima, Axiom or Derive? Personally I find C code
aesthetically unpleasing. If I was to extensively study computer
algebra I would not want to spend my time reading such a low level
language as C. But to each there own.

>> If you want to use C dynamically you are going to need some kind of
>> parser. The parser could be written in C or any other language. It
>> should return to you a tree structure representing the expression
the
>> user typed at the command line. You could then perform some algebra
on
>> it. In this case you may not care if you can overload the plus
>> operator. You could just call it as a C function `pluss(a,b)`, since
>> everything would be done by your custom interpreter (would this
really
>> be C?).
>>

Re: Computer Algebra Algorithms

>

> But the code will be more readable if you can write $a+b$ in your source.

I would hate to see this degenerate as a discussion of infix vs (prefix?). I would consider that such a minor aspect of a programming language. Personally I like infix operators but the Lisp scheme has some advantages. In Lisp An operator/function can be named what every the programmer designers. It can start with numbers consist entirely of numbers or even look like a mathematical expression. In Haskell you can use both infix and prefix form. For instance $+$ is infix but $(+)$ is prefix. Similarly `myfunction` is prefix but ``myfunction`` is infix. The prefix form allows the composition of operators. For instance $(+1).(/2)$ composes the operator divide by two with the operator add one. Since the operators in Lisp are already in prefix form they don't have to be converted before being composed. Anyway, a computer algebra system is another language in itself. It has its own rules and syntax. People generally prefer writing their numeric algorithms in Maple and MATLAB then they do in C. They will only resort to a low level language like C for efficiency or interfacing reasons.

>> If you want to use C and you want to do something useful with CAS I
>> would suggest learning the external function interface of MAPLE.

This

>> will allow you to Manipulate maple data structures in the Maple
>> workspace with C functions. I haven't got into it but perhaps you
>> could even use it to prevent the automatic simplification in Maple.
>> Apparently the best reference for this is the Maple advanced
>> programming guide. It is about 54 dollars.

>>

>

> You can do that with `giac` for free.

Maybe I should take a look at it. There are so many different CAS that I am kind of lost and ask myself where do I start. Like the original poster I am more familiar with C and Java but I want to know lisp because of historic reasons. Early computer algebra was done in lisp based systems. Notably, Axiom, Derive and Mcysma (A.k.A maxima). I think it is important to have a good background of what was done prior and clearly the origin of this begins with Lisp. Lisp is also extensively used in AI (ex. Genetic algorithms) which I am also interested in. Right now I haven't learned much lisp. I read most of a primer but I haven't done any exercises.

I am focusing on learning Maple and Haskell. Maple because it is used via the MATLAB symbolic package and Haskell because it is what a bunch of experts thought lazy pure functional programming language should be. I think learning a computer language is easy and once I learn Haskell the other functional languages (example ML (metta language)) should come quickly. Logical programming might take some time.

Re: Computer Algebra Algorithms

You may wonder if it is worth learning all these languages if you are interested in is CAS. I think that is kind of like is it worth leaning an extensive amount of math when you are interested in engineering. For some it is. I feel, to really understand how to mechanize an intellectual task programming is a strong piece of that. Also a complete computer algebra system should be able to encompass all programming concepts and perform an algebra within a se of programming paradigms. For instance it should no:
 $x-x$ does not equal zero unless we assume that x is not a random number generator.

- ***Follow-Ups:***

- ◆ ***Re: Computer Algebra Algorithms***
◇ *From: Bernard Parisse*

- ***References:***

- ◆ ***Re: Computer Algebra Algorithms***
◇ *From: JohnCreighton_*
- ◆ ***Re: Computer Algebra Algorithms***
◇ *From: Bernard Parisse*

- Prev by Date: ***Re: Any views about an *unmoderated* Mathematica newsgroup?***
- Next by Date: ***Re: Computer Algebra Algorithms lisp vs. C.***
- Previous by thread: ***Re: Computer Algebra Algorithms lisp vs. C.***
- Next by thread: ***Re: Computer Algebra Algorithms***
- Index(es):
 - ◆ ***Date***
 - ◆ ***Thread***