The Marriage of MrMathematica and MzScheme

Chongkai Zhu
mrmathematica@yahoo.com

Abstract

In this paper, I argue that the programming languages provided in current mainstream CASes are not suitable for general purpose programming. To address this problem, I developed MrMathematica. MrMathematica is a connection between Mathematica and PLT-Scheme, which provides the ability to call Mathematica from MzScheme. The two languages share some common ground, but are mostly complementary to each other. MrMathematica enhances Mathematica, and it helps to introduce Scheme to more people (CAS users).

1. Introduction

A Computer Algebra System(CAS) is a type of software package that is used in manipulation of mathematical formulae. The primary goal of a CAS is to automate tedious and sometimes difficult algebraic manipulation tasks. The principal difference between a CAS and a traditional calculator is the ability to deal with equations symbolically rather than numerically. The specific uses and capabilities of these systems vary greatly from one system to another, yet the purpose remains the same: manipulation of symbolic equations. CASes often include facilities for graphing equations and provide a programming language for the user to define his/her own procedures.

CASes began to appear in the early 1970s, and evolved out of research into artificial intelligence (in Lisp), though the fields are now regarded as largely separate. The first popular systems were Reduce, Derive, and Macsyma. The current market leaders are Maple and Mathematica; both are commonly used by research mathematicians, scientists, and engineers.

The programming languages provided in all the current mainstream CASes are not suitable for general purpose programming. To address this problem, I developed MrMathematica, a Scheme based system that keeps the repertoire of Mathematica.

The remainder of this article is organized as follows. Section 2 of this paper discusses why CAS programming language falls and why a real language is needed; Section 3 introduces Mathematica briefly; Section 4 gives details about MrMathematica; Section 5 concludes.

Sixth Workshop on Scheme and Functional Programming. September 24, 2005, Tallinn, Estonia.

2. CAS programmers need a real language

A key issue in the design of CAS is the resolution of what is meant by "evaluation" – of expressions and programs in the embedded programming language of the system.

Roughly speaking, evaluation is a mapping from an object (input) and a specified context or environment to another object that is a simpler or more specific object (output). Example: 2+3 evaluates to 5. More specifically and somewhat pedantically, in a CAS, evaluation involves the conventional programming language mapping of variables or names (e.g. x) to their bound values (e.g. 3), and also the mapping of operators (e.g. +) to their actions. Less conventionally, CAS evaluation generally requires resolution of situations in which a variable "has no value" but stands only for itself, or in which a variable has a value that is "an expression". For example, given a context where x is bound to 3, y has no binding or is used as a "free variable", and z is a+2, a typical CAS would evaluate x+y+z+1 to y+a+5.

In simple cases this model is intuitive for the user and efficiently implemented by a computer. But a system design must also handle cases that are not so simple or intuitive. CAS problem-solving sessions abound in cases where the name and its value(s) in some context(s) must coexist. Sometimes, values are not the only relevant attributes of a name: there may be a declaration of "type" or other auxiliary information. For example it might evaluate $\sin^2 x \leq 1$ to "True" knowing only that x is of type "Real".

CAS builders, either by tradition or specific intent, often impose two criteria on their systems intended for use by a "general" audience. Unfortunately, the two criteria tend to conflict.

1. The notation and semantics of the CAS should correspond closely to "common intuitive usage" in mathematics.

2. The notation and semantics of the CAS should be suitable for algorithmic programming as well as (several levels) of description of mathematical objects, ranging from the abstract to the relatively concrete data representations of a computer system.

The need for this first requirement (intuitiveness) is rarely argued. If programs are going to be helpful to human users in a mathematical context, they must use an appropriate common language. Unfortunately, a careful examination of common usage shows the semantics and notion of mathematics as commonly written is often ambiguous or context dependent. The lack of precision in such mathematics (or alternatively, the dependence of the semantics of mathematical notation on context) is far more prevalent than one might believe. While mathematics allegedly relies on rigor and formality, a formal "automaton" reading the mathematical literature would need to accumulate substantial context or else suffer greatly from the substantial abuse of notation that is, for the most part, totally accepted and even unnoticed by human readers. Consider $\cos(n+1)x \sin nx$.

Because the process of evaluation must make explicit the binding between notation and semantics, the design of the evaluation program must consider these issues centrally. Furthermore, evaluation typically is intertwined with "simplification" of results. Here

again, there is no entirely satisfactory resolution in the symbolic computation programs or literature as to what the "simplest" form of an expression means.

As for the second requirement, the need for programming and data description facilities follows from the simple fact that computer algebra systems are usually "open-ended". It is not possible to build-in a command to anticipate each and every user requirement. Therefore, except for a few simple (or very specific, application-oriented) systems, each CAS provides a language for the user to program algorithms and to convey more detailed specifications of operations of commands. This language must provide a bridge for a computer algebra system user to deal with the notations and semantics of programming as well as mathematics. Often this means including constructions which look like mathematics but have different meanings. For example, in Mathematica x = x+1 is programming language assignment statement; x == x + 1 is an apparently absurd assertion of equality. Furthermore, the programming language must make distinctions between forms of expressions when mathematicians normally do not make such istinctions. As an example, the language must deal with the apparently equal but not identical expressions 2x and x + x.

Programming languages also may have notations of "storage locations" that do not correspond simply to mathematical notations. Changing the meaning (or value) of an expression by a side effect is possible in most systems, and this is rather difficult to explain without recourse to notions like "indirection" and how data is stored. For example, in Mathematica, m[[1,1]]= b assigns value to a position in the matrix m.

With respect to its evaluation strategy, each existing CAS chooses its own twisting pathway, taking large and small sometimes controversial stands on different issues, along the way. Let's see an example in Mathematica:

```
i = 0;
g[x_] := x+i/;i++ > x
```

Or put in Scheme syntax:

```
(begin (Set i 0)
       (SetDelayed (g (Pattern x (Blank)))
                   (Condition (+ x i)
                              (> (Increment i) x))))
```

The two allegedly equivalent expressions (list (g 0) (g 0)) and (Table (g 0) (list 2)) result in (list (g 0) 2) and (list (g 0) (g 0)) respectively.

Other CASes sufferes from similar problems. [4] From the author's own experience, when writing big programs in Mathematica (or some other major CAS), such problems can and will arise, resulting in substantial debugging difficulty.

Providing a context for "all mathematics" without making that unambiguous underpinning explicit is a recipe that ultimately leads to dissatisfaction for sophisticated users.

Is there a way through the morass? A proposal (eloquently championed some time ago by David R. Barton at MIT and more recently at Berkeley) [4] goes something like this: Write in Lisp or similar suitable language and be done with it. This solves the second criterion. As for the first criterion of naturalness – let the mathematician/user learn the language, and make it explicit.

But there is nearly no CA library in Scheme, besides the lightweight JACAL. Statistics shows that for those people who want to do symbolic computation with a computer, nearly all are using a CAS, and nearly none is using Lisp, although most of them also want general purpose programming at the same time. What's worse, CASes that are in/with Lisp (such as MACSYMA, Axiom) have only negligible market share.

So I wrote MrMathematica, which lifts and embeds a popular CAS, Mathematica, into Scheme. Although the currently version targets only MzScheme, its design is portable to any Lisp implementation that can be extended using C. Mathematica was chosen because it has the most dynamic language among major CASes; PLT Scheme was chosen because it has a good interface and a large user group.

3. Introduction to Mathematica

In a typical CAS, an internal evaluation program (`eval` for short), plays a key role in controlling the behavior of the system. Even though `eval` may not be explicitly available for the user to call, it is implicitly involved in much that goes on. Typically, `eval` takes as input the representation of the user commands, program directives, and other "instructions" and combines them with the "state" of the system to provide a result, plus sometimes a change in the "state". Mathematica is one CAS that has a single `eval`.

The central data types of Mathematica are just the same as Scheme: numbers, symbols, and lists. The abstract syntax of the two languages is also congruent: every expression is a list-based tree. To accommodate traditional mathematical expression syntax, Mathematica defines several forms: `InputForm`, `OutputForm`, `TranditionalForm`, `FullForm`, and so on. The FullForm is very close to S-exp, and is the internal representation of expression. A FrontEnd is used to convert between ordinary mathematical expression (InputForm, OutputForm, TranditionalForm) and FullForm.

Mathematica has two major difference compared with Lisp. First, Mathematica doesn't have quote. Second, Mathematica uses array (of pointers) instead of Lisp's linked-list.

The underlying strategy for evaluation in Mathematica is based on the notion that when the user types in an expression, the system should keep applying rules (and function evaluation means rule application in Mathematica) until the expression stops changing. (The example in the previous section just violate this strategy!)

To get a detailed introduction of Mathematica language,please refer to part 2 of [2], or [6].

There are additional evaluation rules for numerical computation in which Accuracy and Precision are carried along with each number. These are intended to automatically keep track of numerical errors in computation.

Besides the rule-based language, Mathematica also offers many mathematical functions and methods, including algebraic manipulation, symbolic calculus, plotting, and so on. Part 3 of [2] describes them in detail.

4. Structure and Interpretation of MrMathematica

Scheme is a meta-language and MzScheme is actually an operation system [3], while Mathematica regards itself only as a scientific computation tool. This determines the architecture of MrMathematica: It works as an extension to MzScheme, which calls Mathematica.

Among all possible interface (between Scheme and Mathematica). I choose to implement the simpest one, MathEval, which is exactly the `eval` used by Mathematica. MathEval is provided as a Scheme function, with input and output done in S-exp, making use of the similarity between S-exp and FullForm. MathEval suffices. Even if you want some "better" interface, the right way to implement it is first to define the same MathEval, and then to define your interface based on it. Another merit of MathEval is that it needs explicit quote, which helps distinguighing between algebra expression and other Scheme value.

Mathematica and MzScheme are both implemented in C, so it is natural for MrMathematica to use C as transmitter. But the ma-

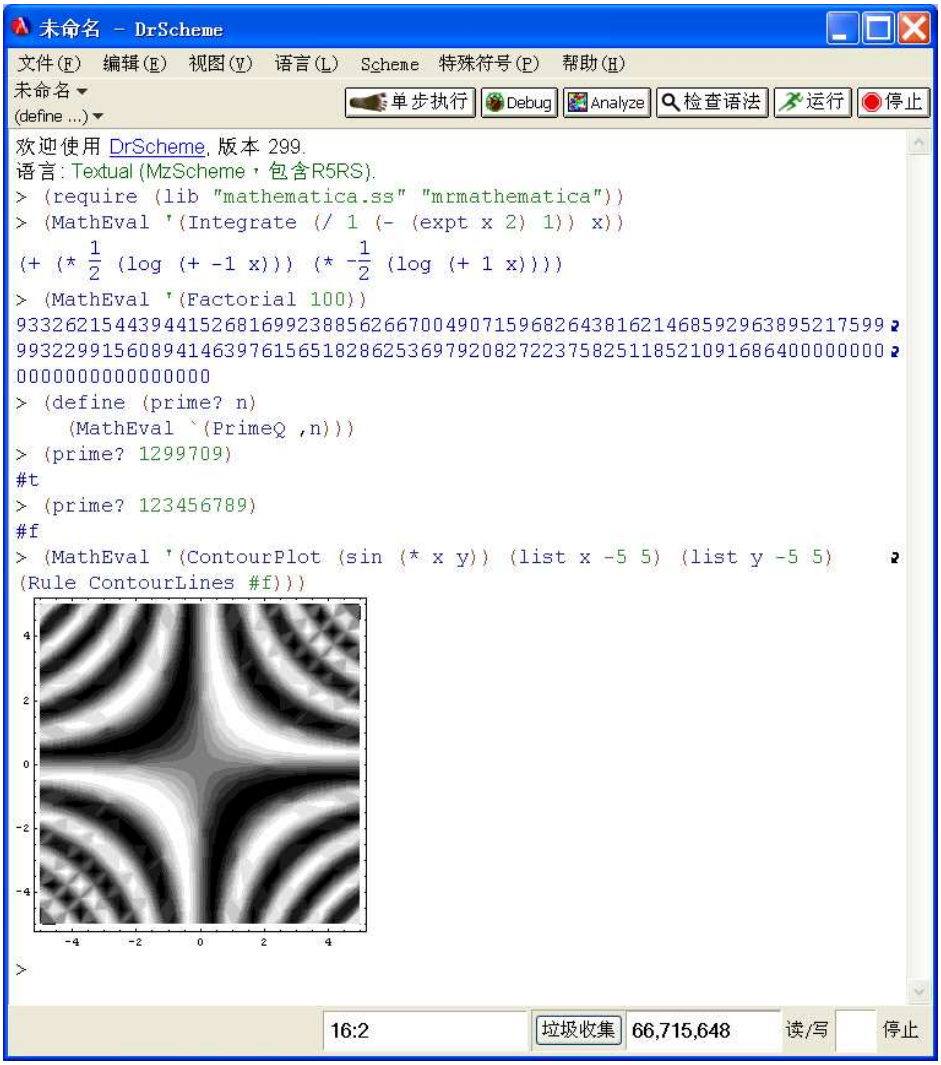

Figure 1. MrMathematica session

jor part of MrMathematica was written not in C but in Scheme. Bottom-up style was used: All needed MathLink (Mathematica's C interface) functions were raised into Scheme in a lower layer implemented as a Scheme module. All the other parts of MrMathematica are written in Scheme, and the final export is the Scheme function MathEval. Compared with the interface provided by MathLink, nearly all the details about the call are encapsulated.

Although the structures of S-exp and Mathematica-expression are similar, the actual keywords are different. The syntax of some pre-defined functions is also distinct. To bridge the gap, I use a separate module in MrMathematica to translate expressions. The result is that a user can write expression just as a Scheme one and send it to Mathematica. In most cases, the output of Mathematica can be directly feed into the Scheme function `eval` or used directly as a Scheme object. The default rules in the translate table are conservative, only dealing with the (exact) common part of Scheme and Mathematica. Programmers can customize the table by new rules.

From the example in Figure 1, we can see that MrMathematica allows every Mathematica Input-Output done in "FullForm" of Mathematica. So CAS users will lose no function from Mathematica, but get the unambiguous, aesthetically appealing, and consistent Scheme. The recommended way to use MrMathematica is, to do all the other programming job in Scheme, and when dealing with mathematical concepts, call the corresponding Mathematica function using MathEval.

You can define your Scheme function that use MathEval, thus using the power of Mathematica with almost no effort. For example, the Mathematica function FactorInteger was raised into Scheme, with exactly the same contract:

```
> (define (factorinteger n)
    (eval (MathEval `(FactorInteger ,n))))
> (factorinteger 111111111111111111)
((3 2) (7 1) (11 1) (13 1) (19 1) (37 1)
 (52579 1) (333667 1))
```

A more efficient version:

```
> (define-syntax factorinteger
    (syntax-rules ()
      ((_ n)
       (map cdr
            (cdr (MathEval `(FactorInteger ,n)))))))
```

For computaion that involves algebra symol(s), explicit quote is used. See the example about integration. To use the return value in Scheme, a explicit call to Scheme's eval is needed:

```
> (define f
    (MathEval
      '(Integrate (/ 1 (+ (expt x 2) 1)) x)))
> f
(atan x)
> (define s (eval `(lambda (x) ,f)))
> (s 0)
0
```

MrMathematica is designed to avoid providing too many features, but also to avoid weaknesses or restrictions. For example, calling multiple or remote Mathematica Kernel(s) is supported; parallel computation is available using PLT's thread utility; variables could all be put in Scheme and the quasi-quote will help transfer their values into Mathematica; Windows, Unix (including Linux), and MacOS are all supported; MrMathematica can render Graphics from Mathematica in DrScheme (this feature needs Scheme and Mathematica running on same machine, which needs further improvement).

Even if your favorite Scheme implementation is not PLT, porting MrMathematica should be easy. There are only three points that are not R5RS and SRFI: the Scheme to C interface, the module system, and the Graphics renderer. MrMathematica uses "Inside MzScheme", the only official C interface for PLT Scheme v20x, as its FFI. As mentioned before, all code that deals with C is in a separate module whose only role is raising C functions. Changing it into different FFI could be done as a routine. The same to module system. To render Graphics from Mathematica in DrScheme, MrEd is used. When using other Scheme implementation, you can easily disable this feature, just as the light-weight version of MrMathematica (designed for MzScheme only instead of full DrScheme) does.

5. Conclusion and Future Work

With MrMathematica, you can use whatever feature you like either from Scheme or from Mathematica. The recommended method to use MrMathematica is to do mathematical compuation in Mathematica and other programming in Scheme. This solves the problem of major CASes: the lack of a good programming language.

Schemers can view MrMathematica as a Computer Algebra library, or a build in term rewriting engine; Mathematica users can view it as a Foreign Language Interface better than that of Java, Perl or Python (string based). After all, the two languages are homologous, thus making the symbiosis.

However, this is only a start of the project. To be really successful, MrMathematica need more applications. Hence this paper. Enjoy hacking with MrMathematica!

For more information about MrMathematica, please visit `http://www.websamba.com/mrmathematica`.

Acknowledgments

Thanks to LinPeng Huang, Matthew Flatt, and Shriram Krishnamurthi for prereading the draft of this paper.

References

[1] PLT Scheme. `http://www.plt-scheme.org/`.

[2] Stephen Wolfram. The Mathematica Book. Wolfram Media, 5th Edition, 2003.

[3] Matthew Flatt, Robert Bruce Findler, Shriram Krishnamurthi, and Matthias Felleisen. Programming Languages as Operating Systems. ICFP 1999.

[4] Richard J. Fateman. Symbolic Mathematics System Evaluators. ISSAC 1996.

[5] Geddes K.O., Czapor Stephen R., and Labahn George. Algorithms for Computer Algebra.Kluwer Academic, 1992.

[6] John Gray. Mastering Mathematica. Academic Press,Inc, 1994.

[7] G J Chaitin. Algorithmic Information Theory. Cambridge University Press, 2004.

[8] Stephen Wolfram. A New Kind of Science. Wolfram Media, 2002.

[9] Olin Shivers. A Scheme Shell. `http://www.scsh.net/docu/scsh-paper/scsh-paper.html`

[10] Aubrey Jaffer. Jacal. `http://swissnet.ai.mit.edu/jaffer/JACAL.html`

[11] Maxima. `http://maxima.sourceforge.net/`

[12] Axiom. `http://savannah.nongnu.org/projects/axiom`

[13] Reduce. `http://www.reduce-algebra.com/`

[14] Mapple. `http://www.maplesoft.com/`