Towards Mathix / Math-x, a computer algebra language that can create its own operating system, and that is amazingly user-friendly and secure

Thomas Colignatus, August 19 2009, some clarifications August 23 2009

http://www.dataweb.nl/~cool

Summary

Given the current supply and demand for computer languages and systems, there is a clear window of opportunity for the software community to collaborate and create Mathix and Math-x. The base would be Oberon, which itself derives from Algol / Pascal / Modula and which is better designed than Unix / Linux, and which now has the system Bluebottle / A2 at http://bluebottle.ethz.ch. Mathix is defined as Oberon/A2 extended with a suitable computer algebra language (say, CAL). Math-x is defined as Minix ported to Oberon/A2 (via Cyclone) and extended with that CAL. Mathix and Math-x thus can create their own OS. Mathix and Math-x are flexible in that they contain both a strongly typed dialect relevant for OS creation (current Oberon) and a more flexible dialect relevant for the human user (CAL, to be created). The dialects have much syntax in common but the CAL allows more flexibility. The CAL is internally translated into Oberon/A2 which allows compilation as well.

Note

This paper is best understood in the context of my book *Elegance with Substance* on mathematics education - see http://www.dataweb.nl/~cool/Papers/Math/Index.html. My book advises that each national parliament investigates the stagnation in doing mathematics on the computer in school. This paper suggests how the software community might anticipate on common sense conclusions and start working on improvement.

This document has been created in *Mathematica* 7. This program allows the inclusion of hypertext (clickable but hidden links) and transformation into PDF but those PDF (either created directly or via printing via PrimoPDF) apparently have the links removed (or they are not shown in Foxit). It is an option to save from *Mathematica* in HTML format, but, curiously, such a file comes as a collection of files and links and not as a single file whence it is not easily transportable. It is defeat, but it remains the best approach to include all links *verbatim*.

■ Introduction

We start out with a quest. Considerations then cause a choice. For the presentation it is clearest to first present the choice and then elucidate on the considerations. We can close with the consequences.

■ The quest

In our advanced computer age in the year 2009, we would like to have a computer algebra language that not only allows us to do mathematics on the computer, but that ought to be capable to compile itself and to create the operating system (OS). The advantage is that code becomes both more rigorous and transparant. Teaching, learning and use become more agreeable and effective. An advantage is that more complicated mathematical software would also become directly available for use in the OS and thus contribute to speed, security and user friendliness. Regrettably, we don't have that yet. It should be free software as well of course.

We are all aware of de abundance of software, see the somewhat phyrric http://rosettacode.org/wiki/Main_Page. The idea of this present paper is to simplify all of this to *a basic system with a uniform syntax for the standard applications*.

A direct refutation

An expert directly stated: “A disk driver is 50 pages of code that simply stuff bits into registers to make the disk work and then manage the data that comes in and verifies checksums. There aren’t any complex algorithms. It is all grunt work. So I really don’t believe a math-type language has much to offer.”

I stand corrected. However, a mathematician sees math everywhere. The disk might be seen as a database and algorithms on the database might warrant speed and security. A bottom line remains that the syntaxes of most OS’s and their sources differ from one another. A gut feeling is that there is a powerful language and environment out there.

Users of Windows will spend a decent amount of their time to updates and checking for virusses. The load for users of Linux is less but potentially it can increase when more people adopt Linux. Thus for Linux we would wish to have a well structured and secure alternative, best formulated in that new computer language.

■ The choice

Oberon is a language that creates its own OS - see http://www.oberon.ethz.ch and http://en.wikipedia.org/wiki/Oberon_operating_system. In this paper "Oberon" will be a container word but sometimes it may be useful to distinguish the language seen as language itself and the implementation into the actual operating system Bluebottle / A2 at http://bluebottle.ethz.ch. See JAOS at http://www.bbos.org/ethmirror/bluebottle.ethz.ch/jaos/index.html. Its community looks into computer algebra, see Gruntz & Weck, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.6614. Oberon is free software equivalent to the GNU PL - see ftp://ftp.inf.ethz.ch/pub/software/Oberon/ETHOberon/license.txt;type=a.

It remains to extend Oberon (thus, Oberon/A2) with computer algebra, creating **Mathix**. Then Mathix will have two sub-languages or dialects, (1) the Oberon current language that is the vessel towards compilation and OS, and (2) the computer algebra language that is easier for the human user (e.g. by allowing a free type). Perhaps we can compare this to the French and Anglo-Saxon roots in the modern English language, where different expressions are possible.

Minix has a different approach from Linux and Oberon - see http://en.wikipedia.org/wiki/Microkernel, http://www.minix3.org, http://www.cs.vu.nl/~ast/jobs/R3S3-summary.pdf and see the license that is equivalent to the GNU PL http://www.minix3.org/license.html. For OS design, it would seem to be obvious that Minix has the future. Since it has been written in C we would need to port C to Cyclone to Oberon, and then the ported Minix code can be extended with computer algebra, creating **Math-x**.

■ How that choice came about

(1) General criteria

The commercial combinations of Maple / *Mathematica* on Windows / Mac / Unix allow ease of use and power of application. Their cost need not be overrated if they are used frequently so that the cost per minute decreases. But they create intellectually artifical boundaries when a user wants to change that software. It is not self-evident how software can be integrated. Formats differ. Editting in *Mathematica* differs from editting in Word. The free statistical sofware package R has its own implementation language but this is far from a computer algebra language and one wonders why the computer algebra language is not standardly available from the operating system up.

It would seem to be a clear argument that a good computer language also ought to be able to compile itself and install itself on a computer. In practice we meet specialisation and language barriers. Added to the financial costs there are learning and testing times.

One might argue that Unix / Linux and C are so married to each other that they jointly form a language that compiles itself. However, there are differences. For example, comments in C have the form /* ... */ while Bash programs in Linux have #... (till the end of the line).

(2) The jungle

For the OS there is the choice between Windows, Apple OS X, Unix, Linux, Minix, Oberon, and all their versions, and possibly some that I overlooked or don't feel like mentioning. For languages we can consider Lisp, Fortran, C, Haskell, Python, Oberon, etcetera. A comparison of computer algebra systems can be found at http://en.wikipedia.org/wiki/Comparison_of_computer_algebra_systems.

(3) Considerations

Since we are considering free software the possibilities for the OS reduce to Linux, Minix and Oberon. Speed and garbage collection are important here.

Tanenbaum clarifies the crucial distinction between Linux and Minix. This distinction would also hold for Oberon versus Minix. For stability and security it would seem to be obvious that Minix has the future. Minix however has been written in C but Tanenbaum requires more structure and then mentions to consider Cyclone. This is a variant of C that tries to become well-structured, see http://cyclone.thelanguage.org and http://en.wikipedia.org/wiki/Cyclone_(programming_language).

Oberon has one of the strongest possible foundations in computer theory - see the genealogy from Algol, Pascal and Modula. Oberon seems to have lost out in the onslaught in the USA where commercial competition and the fear for litigation have dominated developments. Fortunately the free software movement - many academics and http://www.fsf.org - has defended freedom but it is striking how US developers (including FSF) are focussed on Unix and C. There clearly is the “lock in” of an established base.

For computer algebra, Sage is the best available free language, and itself combines Maxima and Python, see http://www.sagemath.org, http://maxima.sourceforge.net and http://www.python.org. It is somewhat curious that Maxima has been developed (as a superior language) in Lisp, that Lisp has a compiler, but that Maxima has no compiler (but perhaps I overlooked a program that converts Maxima to Lisp again). Haskell for functional programming can be mentioned too, see http://www.haskell.org and http://en.wikipedia.org/wiki/Haskell_(programming_language).

It might be mentioned that Google has the same distaste for virusses and security updates and works on Chrome OS, see http://en.wikipedia.org/wiki/Google_Chrome_OS and http://googleblog.blogspot.com/2009/07/introducing-google-chrome-os.html. However, for math applications it does not seem advisable to rely on cloud computing (only).

■ Minor consequences

(1) For C, Linux and Minix

We need ports (source to source translators) from C to Cyclone and then to Oberon. In that way Linux and Minix source codes can be ported. Since Minix has much less code than a convential Linux distribution, Minix can make the quickest transfer. When Math-x exists, it may well get so much momentum that Mathix need not arise (since Minix has the future).

(2) For List and Maxima

We need a port from Lisp to Oberon. Then Maxima becomes available in Oberon. (Optionally Punimax ftp://ftp.gwdg.de/pub/linux/math/punimax.) Note that Oberon compiles so that Maxima's Lisp source ported to Oberon compiles by implication. However Maxima expressions themselves need not compile yet.

(3) For Sage and Python

As Sage uses Maxima (Lisp) and Python (C), it can be ported in above ways. The Python language compares to some extent to computer algebra languages and possibly its system amounts to an alternative interpreter. Currently, there is work on a compiler for Python and this C program might be ported too. Sage relies on other programs as well which will require further ports.

(4) TeXmacs, R, Axiom

Importantly, we would want to port TeXmacs - http://www.texmacs.org - as well, requiring a conversion of Lisp / Scheme to Oberon. Likely the TeXmacs instruction language must be transformed into Math-x as well.

Similarly for R and the idea to include a computer algebra system into R - http://www.r-project.org, http://faculty.ucr.edu/~tgirke/Documents/R_BioCond/R_Programming.html (C and R itself) and http://www.math.aau.dk/~slb/kurser/software/RCompAlg.pdf.

Similarly for Axiom - http://www.axiom-developer.org/ (Lisp and C++).

Since Math-x is the higher order computer algebra language, it is conceivable that programs like TeXmacs and R are ported to Math-x instead of Oberon. This depends upon how fast the computer algebra dialect (CAL, to be made) becomes available and how urgent the use of the programs is.

(5) Zonnon

There is the initiative of Zonnon, an object oriented language in Oberon with an innovative design, http://www.zonnon.ethz.ch. It uses the .NET framework and at this moment it is not clear to me how it installs in the free software environment. We would like to see Zonnon available as a dialect in Oberon. Zonnon helps to clarify that an OS actually consists of levels of OS. There is basic Grub, then core system functionality, then more flexible system management. Perhaps more stages. All would come with a consistent syntax but there can be dialects to allow for different programming styles.

(6) BIOS and Grub

Grub already comes with minimal recognition of keyboard and screen. Once such functionality exists it seem logical, perhaps only rationally and not practically, to ask whether Grub isn't actually some kind of an OS. We would like to transform it and extend it with our supposed Math-x, to be created. Likely the best answer would be that Grub would also be defined in Math-x, though uses only a minimal core and then relinquish control to any other OS of preference. Rephrased differently, we likely would want Math-x to be equipped with such a grub feature.

■ Major consequence

The major consequence is that we must try to settle on a computer algebra language (syntax). We have syntax of *Mathematica*, Maple, Maxima, Python, Axiom and Haskell. The best must be selected, which gives the CAL to be made. The existing CAs can be ported to CAL.

■ Design principles

The basic design is Minix in Oberon extended with the CAL to be made. However, on the way, it would be advisable to prevent errors made in current designs.

For the human interface, it would be advisable to use insights from psychology, see

Verhoef at http://www.humanefficiency.nl/gui/multi_dimensional_interfaces.shtml

Also, as Math-x is targetted to be used in mathematics education (and applied in physics, economics and biology education too since these use math too) it is advisable to be aware of 'evidence based education'. For example, it is my conjecture that 2 + 1/2 is the proper mathematical notation of two and a half, instead of the common $2\frac{1}{2}$ (that looks like multiplication). Readers of *Elegance with Substance* may agree. Conceivably it suffices when mathematicians agree on this. However it will be useful to perform tests when feasible and to maintain that philosophy.

■ How to proceed

Focus

The programming world seems focussed on .NET, C++ and Python. However, the issue of security will help to focus attention. It will help that Mathix / Math-x will have features of ease as in Python.

Organisation

Clearly, Computer Science departments at the universities and institutes of technology would have to cooperate, perhaps in the Algol fashion.

There can be a project / org that adheres to the FSF conventions to provide overall guidance. One option is that computer users in the world donate to the Free Software Foundation that then buys up creations to put them into the GNU PL (in the fashion how Microsoft has grown).

In anticipation, www.math-x.org has been reserved by the HACC foundation in The Hague, see http://pir.org/index.php?mode=whois&db=content%2FWebsite&tbl=Registrants&id=3&whois_query_field=math-x. It appears that Mathix.org already has been used but this does not matter.

Preferably, there will be a constitution, user parliament, court, executive, with rules for decisions and operations. It is not clear whether these differ for Math-x.org or FSF.

Things to prevent

Academic initiatives can be highly ineffective and increase the Babylonic chaos. The histories of some commercial companies around Macsyma, Axiom and Aldor are somewhat depressing but ventures as Maple and *Mathematica* are encouraging for their stability and product quality. Some ventures apparently are able to pick the cherries from the academia and let the seeds grow and bloom again. It is worth an effort to copy that model with free software.

There are curious phenomena. One of the features of Haskell is lazy programming but this is available in *Mathematica*. Similarly for Typing and lambda calculus. Why don't they use *Mathematica* ? The main argument for the creation of Haskell would be an open source development and an academic check on rigour but with the added cost of another syntax. Haskell doesn't come with converters from commercial packages to Haskell. Users who already had been forced to a commercial environment are left to themselves again, especially when they would like to combine some aspects of Haskell with other aspects still not provided for by Haskell. All this would be solved by Math-x, though with another syntax.

Some principles

Some principles thus would be:

* We will never be able to arrange everything to everyone's full satisfaction.

* Legalese should be kept to a bare minimum.

* Academics should do their thing for the academia, but can help in the public domain, and when they help then they should focus on helping out instead of doing their thing.

* Free software runs on donations. Donations run on good free software.

■ A note on *Mathematica*

I have only hands-on experience with *Mathematica* (and Algol, Pascal and Fortran). I am in favour of using "=" for identity / equality, ":=" for direct assignment and "=:" for delayed / lazy assignment, while *Mathematica* uses "==" for identity / equality, "=" for direct assignment and ":=" for delayed / lazy assignment. For me, [*a, b, c*] can be an ordered list and {*a, b, c*} an unordered list (set), while *Mathematica* uses the last for both though with different Attributes. I am in favour of a conditional list, such as Probability[*a, ..., b,* Given, *c, ..., d*] (where ";" and "|" cannot be used, and with an orderless form). Thus I am not hung up on the *Mathematica* language. I am in favour of adequate conversion from the *Mathematica* language to the CAL to be made. PM. There is distinction between the *Mathematica* language and the executable that evaluates it. Mathematics and its notation cannot be copyrighted but software can be proprietary.

Relevant in this context is Niklaus Wirth, http://www.inf.ethz.ch/personal/wirth/Articles/GoodIdeas_origFig.pdf

While *Mathematica* is a computer algebra system - more precisely "a system for doing mathematics on the computer" - and is quite flexible in its use, it might be called to attention that it also allows more rigorous programming. For example, *f*[x_Integer] specifies that *f* accepts only integers. There are Begin and End statements (for a Context, that can be taken as a program). We can flexibly extend a function e.g. by including *f*[x_String] and also work interactively in a Context (since programs can be interactive). Thus, *Mathematica* is already much of our Math-x but it cannot compile itself and be used as an OS. It would be great when WRI, the maker of *Mathematica,* would turn into a public utility company so that we could all work towards such goals. We may speculate what WRI is up to but it seems best to work towards Math-x anyhow.

Given my background and given the goal of arriving at a new computer algebra language, it seems a good idea that I comment on Richard Fateman's "Review of *Mathematica*" - http://www.cs.berkeley.edu/~fateman/papers/mma.review.pdf. On his website (page retrieved 2009-08-17) he states (undated, likely a bit after 2002): "Although it is now a decade old (and is a review of a computer program that continues to evolve), I think a careful reading shows that the broad comments are still valid."

Since my review of Fateman's Review will be lengthy, I put that into another paper.

PM 1. Richard Fateman - http://www.cs.berkeley.edu/~fateman/ - is one of the creators of Macsyma (Maxima's precursor) and thus one of the founding fathers of computer

algebra. Macsyma indeed is a source of inspiration of *Mathematica* itself, see e.g. http://en.wikipedia.org/wiki/Symbolic_Manipulation_Program.

PM 2. Advised reading from Fateman's website are:

● 2001 *criticism of the OpenMath project* http://www.cs.berkeley.edu/~fateman/papers/openmathcrit.pdf. For example "The fact that binding, evaluation, substitution and related concepts have been so mangled by computer algebra systems (for example Maple and Mathematica) should have served as object lessons to the OpenMath designers. Instead of standing on the shoulders of those who have gone before, we are standing on their toes."

● 2000 *Software Fault Prevention by Language Choice: Why C is Not my Favorite Language* http://www.cs.berkeley.edu/~fateman/papers/software.pdf

●1999 *Problem Solving Environments and Symbolic Computation* http://www.cs.berkeley.edu/~fateman/papers/pse.pdf

● 1999 *Mock Mathematica* www.cs.berkeley.edu/~fateman/mma1.6.tar.gz is a Lisp program to evaluate expressions in the *Mathematica* language. If Lisp can be ported to Oberon then this might be a starting point to port *Mathematica* programs to Oberon or Math-x as well.

●1996-99 *Symbolic Mathematics System Evaluators* http://www.cs.berkeley.edu/~fateman/papers/eval.ps. "Perhaps if there is a lesson to be learned from the activity of the last few decades, it is this: For computer scientists to provide at one fell swoop a natural notation and evaluation scheme for all mathematicians and mathematics is both overly ambitious and unnecessary." Indeed, we should only target for a uniform syntax for the standard applications, usable in K12 and up.

■ Conclusion

The book *Elegance with Substance* already showed the need for a open source computer algebra system in an open source computer environment. By implication, but made explicit here, it is merely logical to require that the system can compile itself and be its own OS. A print command in the OS would best have the same syntax as one in the application. Kids in school benefit from a stable and uniform environment, though we also want some advancement (and get rid of the clay tablets). The current jungle is not the right answer and we can create a system with a uniform syntax for the standard applications.