Modern Mechanized Mathematics

Jacques Carette

McMaster University

Joint Lab Meeting – Friday May 29th, 2009

joint work with William M. Farmer

Overview

Categories of Tools for Mathematics

Note that each category also contains many small specialized systems.

- Numeric
  - Statistics – SPSS, S-Plus, R
  - The rest (calculus, linear algebra, ...) – Matlab, Scilab
- Symbolic
  - Computation – Mathematica, Maple, MuPAD, SAGE, Axiom, Aldor
  - Proofs – Coq, Isabelle, Mizar, PVS, HOL, IMPS, Ωmega, ...
  - Both – Theorema, Focalize, HOL-Maple, PVS-Maple, MathScheme

Rough user count from various web sources:

- Stats 100-200 million
- Matlab 30-40 million
- Computer Algebra 5-6 million
- Theorem Provers (including SAT & Model checking) 400,000.
- Interactive Theorem Provers 30,000-40,000.

Categories of Tools for Mathematics

Note that each category also contains many small specialized systems.

- Numeric
  - Statistics – SPSS, S-Plus, R
  - The rest (calculus, linear algebra, ...) – Matlab, Scilab
- Symbolic
  - Computation – Mathematica, Maple, MuPAD, SAGE, Axiom, Aldor
  - Proofs – Coq, Isabelle, Mizar, PVS, HOL, IMPS, Ωmega, ...
  - Both – Theorema, Focalize, HOL-Maple, PVS-Maple, MathScheme

Rough user count from various web sources:

- Stats 100-200 million
- Matlab 30-40 million
- Computer Algebra 5-6 million
- Theorem Provers (including SAT & Model checking) 400,000.
- Interactive Theorem Provers 30,000-40,000.

Categories of Tools for Mathematics

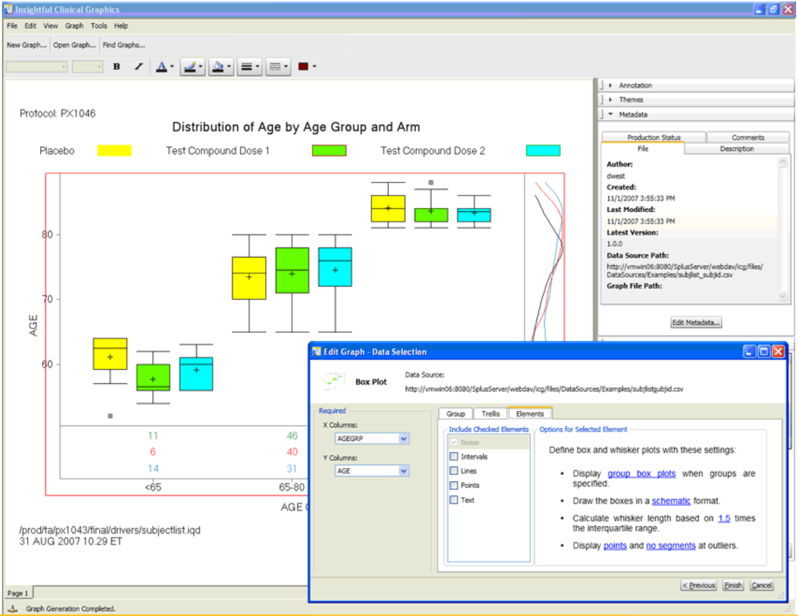
Note that each category also contains many small specialized systems.

- Numeric
  - Statistics – SPSS, S-Plus, R
  - The rest (calculus, linear algebra, ...) – Matlab, Scilab
- Symbolic
  - Computation – Mathematica, Maple, MuPAD, SAGE, Axiom, Aldor
  - Proofs – Coq, Isabelle, Mizar, PVS, HOL, IMPS, Ωmega, ...
  - Both – Theorema, Focalize, HOL-Maple, PVS-Maple, MathScheme

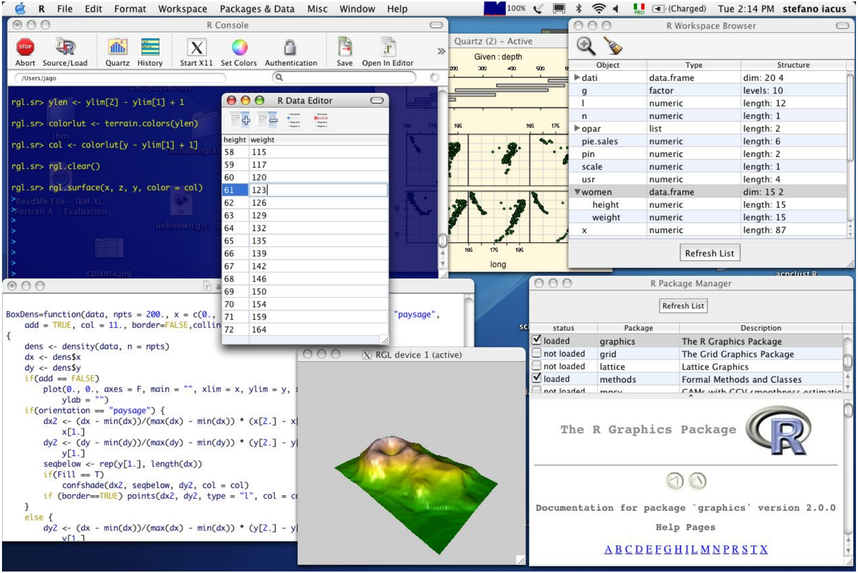
Rough user count from various web sources:

- Stats 100-200 million
- Matlab 30-40 million
- Computer Algebra 5-6 million
- Theorem Provers (including SAT & Model checking) 400,000.
- Interactive Theorem Provers 30,000-40,000.

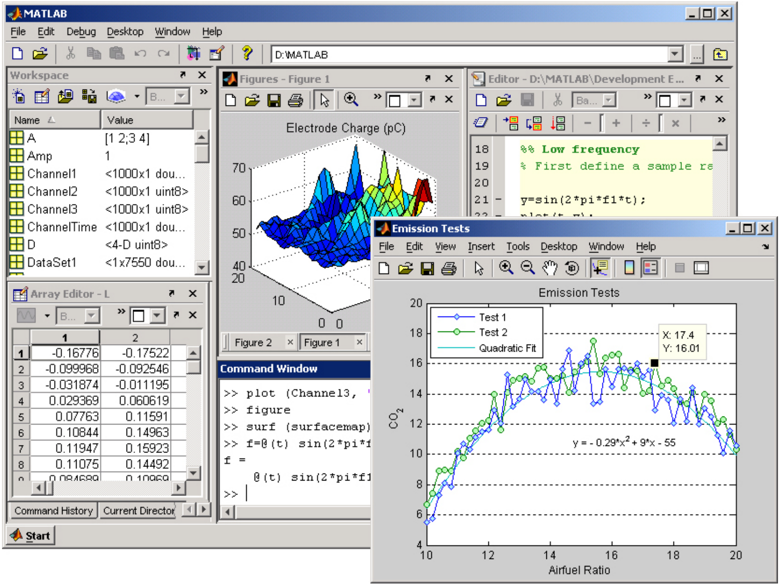
And your experience may look like



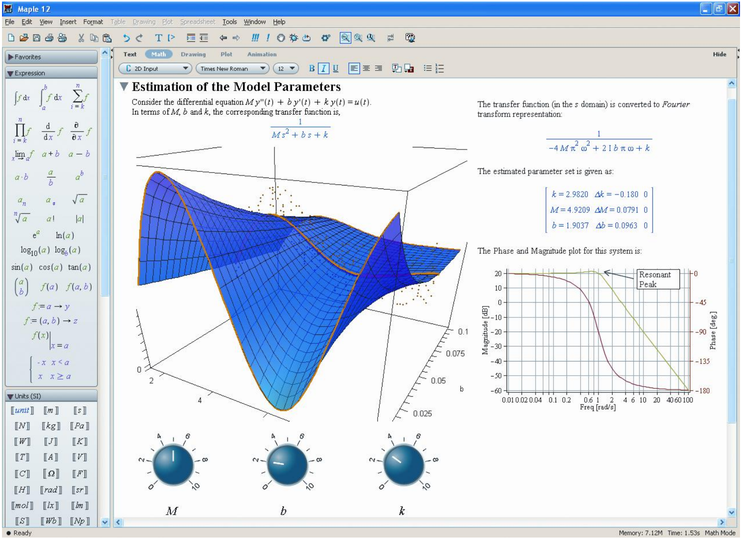
And your experience may look like



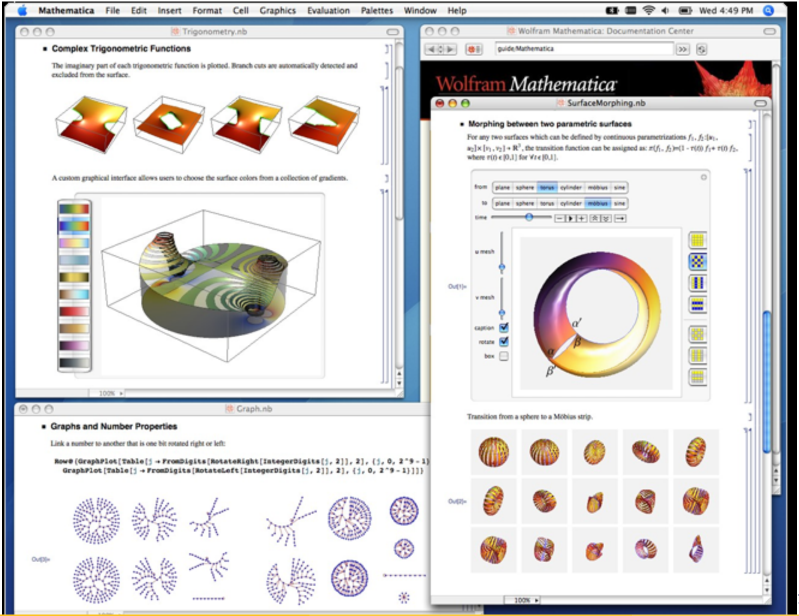
And your experience may look like



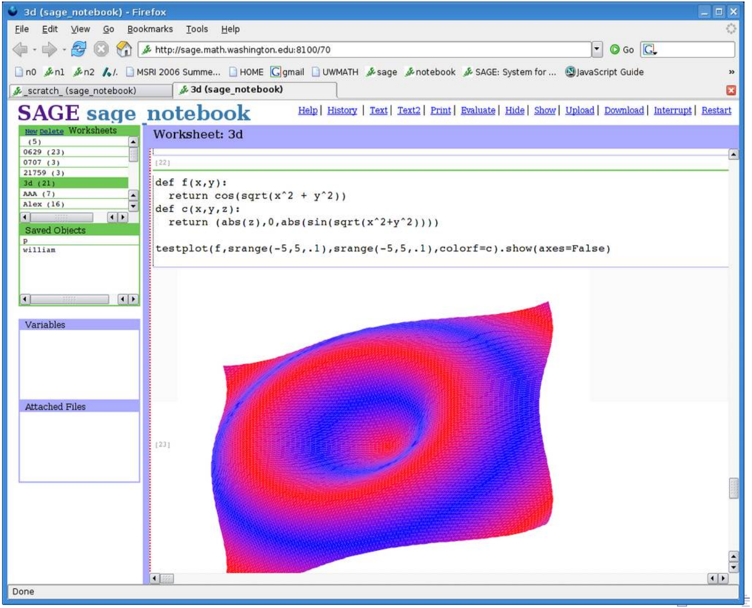
And your experience may look like



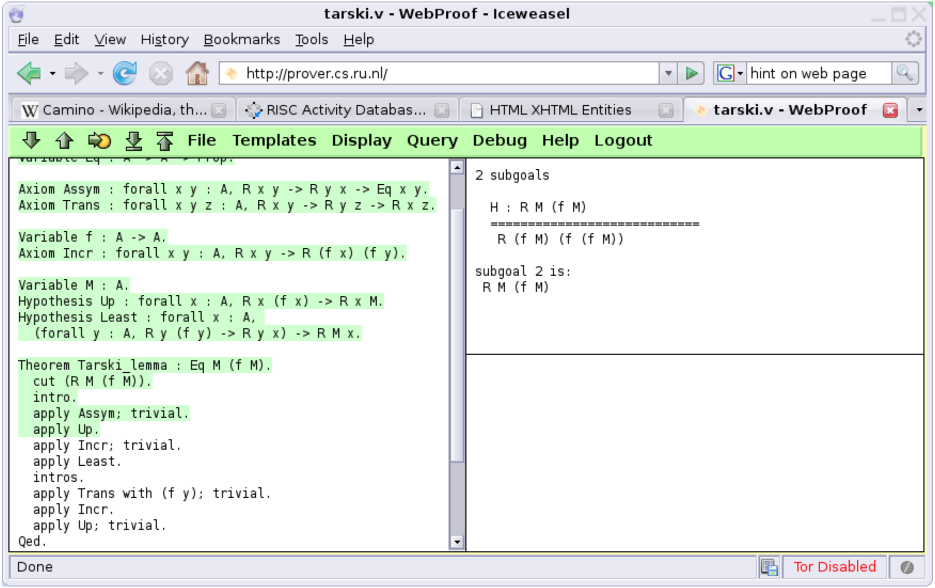
And your experience may look like



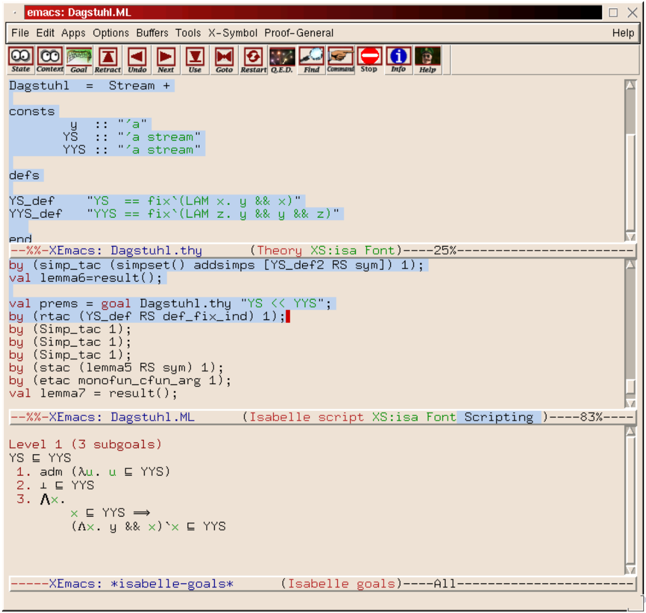
And your experience may look like



And your experience may look like



And your experience may look like



Pre-history

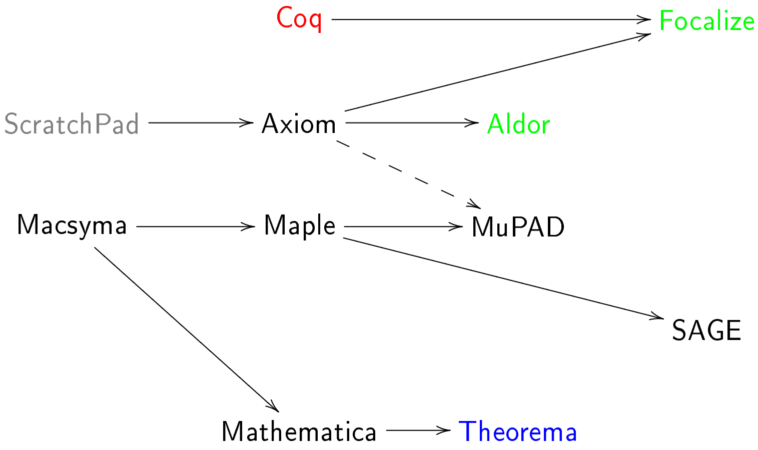
Interactive Theorem Provers:

1968	Automath	Netherlands, de Bruijn
1971	nqthm	US, Boyer & Moore
1972	LCF	UK, Milner
1973	Mizar	Poland, Trybulec

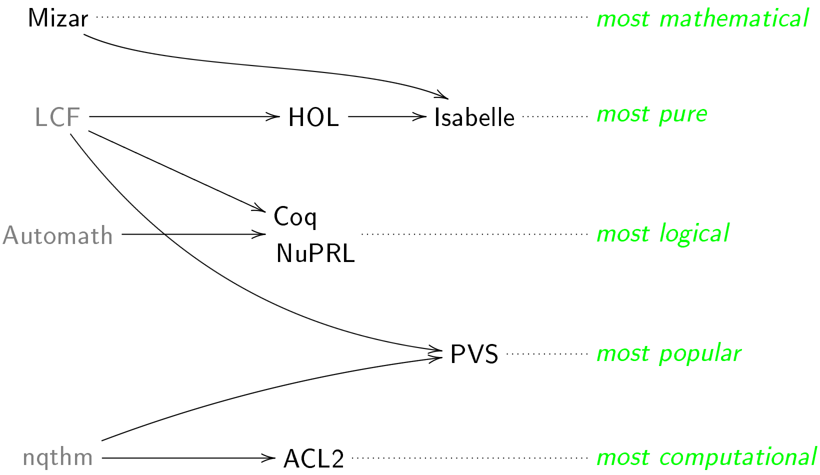
Computer Algebra:

1963	Schoonship	Netherlands, Veltman
1965	Formula Algol	US, Perlis
1967	Macsyma	US, Engelman & Moses
1968	Reduce	UK, Hearn
1968	ALTRAN	US, Hall
1971	ScratchPad	US, Jenks

A short history – Computer Algebra



A short history – (Interactive) Theorem Proving



slide courtesy of Freek Wiedijk

Calculemus Network - `www.calculemus.net`

Calculemus! (Let us Calculate)

Gottfried Wilhelm Leibniz

Mission

The CALCULEMUS interest group is a loosely coupled network of research groups and individuals interested in joining forces for he design of a new generation of mathematical software systems and computer-aided verification tools based on the integration of the deduction and the computational power of deduction systems and computer algebra systems respectively.

Meetings since 1998, generally alternating between deduction and "computation" conferences.

The Mathematics Process

In mathematics, we

- define new concepts, define new notations
- state, in convenient ways, problems to be solved
- conduct experiments
- make conjectures
- prove theorems
- gain *insight* through proofs, computations and visualization
- turn theorems into algorithms
- communicate our results
- reuse previous results

Our goal: build a tool that helps us do all of that.

Current tools

Current (oversimplified)

	TP	CA
Orientation	semantics	results
Mathematics	abstract	concrete
Steps	deductive	computational
Rigor	high	low
Context	explicit, axiomatic	implicit, algorithmic
Speed	very slow	fast
Ease of use	low	medium-high

Challenges

1. Put computation and proof on equal footing.
   - Proofs need computations (Poincaré principle), and
   - computations need proofs (side-conditions)
2. Symbolic computation is on syntax but about semantics
3. Efficiency matters. Correctness matters.
4. Ease of use is important. Proofs should be precise.
5. Modularity. Humans work in very rich theories.
6. Small team. Huge task (~ 150 person-years).
7. Concrete representations. Sufficient abstraction.
   - sparse polynomial data-structure. ${}_2F_1\left(1/2, -3;\ 1;\ x\right)$ is a polynomial.

Solution pieces: Biform theories

Issue

Computation and proof on equal footing.

Definition

A Biform theory consists of *axioms*, *algorithms* and *meaning formulas* linking axioms and algorithms.

Intuition for programmers: specifications and programs.

A "step" in a development can be either from a deduction rule or the application of an algorithm. These are encapsulated in transformers.

Constructively $\Rightarrow$ program extraction. But also allows *black boxes*.

Solution pieces: Chiron

Issue

Symbolic computation is on syntax but about semantics.

Chiron is a typed logic based on von Neumann-Bernays-Gödel (NBG) set theory, with facilities to reason about undefinedness as well as the syntax of expressions.

Intuition: a typed logic and a dependently-typed lambda-calculus with `eval` and `quote`.

Lots of convenient features, including *definite* and *indefinite description*, *undefined* as well as *non-denoting* expressions. β-reduction is definable in the logic.

Allows computation (and proof) with objects whose denotations are non-constructive (eg: algorithm for normal-form of piecewise-defined functions over classical $\mathbb{R}$)

Solution pieces: Genericity and Generativity

Issues

Efficiency matters. Correctness matters.
Concrete representations. Sufficient abstraction.

Intuition: `C++` templates done right.

Mathematics is full of generic concepts and algorithms which can be proven "correct" once and for all. However, efficiency depends crucially on concrete representations. The trick: correctly specializing generic algorithms to get efficiency for concrete cases.

The tools: typed metaprogramming and partial evaluation.

Solution pieces: Context and Automation

Issue

Ease of use is important. Proofs should be precise.

Context is used pervasively by humans when *doing* mathematics.

- Always reason in a "local context"
- Make specifying context easy (eg: Calculus, Circuit Design, ...)
- Domain Specific Languages everywhere!
- Automate everything that can be (eg: type inference, definedness guards, simplification, etc)

Note: Type inference is undecidable in Chiron. Use abstract interpretation to obtain as good an approximation as possible.
Use partial evaluation (on theories!) to remove most definedness guards automatically upon specialization.

Solution pieces: Little Theories & High Level Theories

Issue

Modularity. Humans work in very rich theories.

The Little Theories idea is to assemble larger theories out of small components – modularity taken to an extreme. Note that mathematics is usually presented (in textbooks) in this manner.

A High Level Theory (HLT) is a very rich theory (think 800 page textbook on calculus) in which it is convenient to work.

Intuition: Mathematics is naturally a giant network of definitions, theorems and algorithms (little theories). Doing mathematics in a HLT hides the network and provides an amazing IDE.

The theory network is the developer's view while the HLTs are the end user's view.

Solution Pieces: Trustable Communication

Issue

Small team. Huge task ($\sim$ 150 person-years).

Idea: Use translations and interpretations between theories to transport results from one system to another.

Intuition: Services! Build system out of (trustable) components.

A service is formally a transformer in a biform theory. A result is a theorem. Eg: `plus` $: \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$ is a simple transformer, which can be used as `plus(5,6)` and will return the theorem $5 + 6 = 11$.

Technically requires that services be formally specified. Interpretations can be very difficult to construct.

The "solution pieces" of the preceding 6 slides correspond to Publication(s) of J. Carette and/or W.M. Farmer & co-authors.

Applications

Mathematics has applications everywhere, but we feel that mechanized mathematics can bring the biggest benefit to:

- Software development – Software Certification
- Model-based development in science and engineering
  - eg: parametric PDE + objective function to optimize
- Building a digital mathematics library
- Mathematics education (via automated TA and interactive lab)

Note: mathematics research is purposefully absent from this list.

MathScheme 2.0

> *Machines should work. People should think.*
>
> *Richard Hamming*

Long term goal: mechanize the mathematics process.

Short term goal: integrate theorem proving and computer algebra

We need a new system because all current systems:

1. have chosen (in their fundamental design) to tackle only some of the challenges, and
2. are being improved only incrementally.

A Biform Theory, using Chiron

$\texttt{Theory Derivative-Real1D \{}$

$\quad \mathsf{derivative} : (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$

$\quad \texttt{axiom}\ \forall f : (\mathbb{R} \to \mathbb{R}). \forall x : \mathbb{R}.$

$$\mathsf{derivative}(f)(x) \simeq \lim_{\epsilon \to 0} \frac{|f(x+\epsilon) - f(x)|}{\epsilon}$$

$\quad \mathsf{diff} : E_{(\mathbb{R} \to \mathbb{R})} \to E_{(\mathbb{R} \to \mathbb{R})}$

$\quad \texttt{meaning}\ \forall f : E_{(\mathbb{R} \to \mathbb{R})}. [\![\mathsf{diff}(f)]\!] \simeq \mathsf{derivative}([\![f]\!])$

$\}$

If you wanted to use Maple's diff here, you would have to change that meaning formula to

$$\forall f : E_{(\mathbb{R} \to \mathbb{R})}. (\mathsf{total}(f) \wedge \mathsf{differentiable}(f)) \Rightarrow ([\![\mathsf{diff}(f)]\!] \simeq \mathsf{derivative}([\![f]\!]))$$

A Biform Theory, using Chiron

```
Theory Derivative-Real1D {
```

derivative : $(\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$

`axiom` $\forall f : (\mathbb{R} \to \mathbb{R}). \forall x : \mathbb{R}.$

$$\mathrm{derivative}(f)(x) \simeq \lim_{\epsilon \to 0} \frac{|f(x+\epsilon) - f(x)|}{\epsilon}$$

diff : $E_{(\mathbb{R} \to \mathbb{R})} \to E_{(\mathbb{R} \to \mathbb{R})}$

`meaning` $\forall f : E_{(\mathbb{R} \to \mathbb{R})}. [\![\mathrm{diff}(f)]\!] \simeq \mathrm{derivative}([\![f]\!])$

```
}
```

If you wanted to use Maple's diff here, you would have to change that meaning formula to

$$\forall f : E_{(\mathbb{R} \to \mathbb{R})}. (\mathrm{total}(f) \wedge \mathrm{differentiable}(f)) \Rightarrow ([\![\mathrm{diff}(f)]\!] \simeq \mathrm{derivative}([\![f]\!]))$$

Code Generation - algorithm families

Encode "design concepts" present in a "software product line" composed of variants of an algorithm.
Case study – Gaussian Elimination & LU Decomposition. Rationale: found 80 different implementations in Maple's library.
Result: generated code is *identical* to human-written versions.

```
module GVCI = GenericVectorContainer(IntegerDomainL)
module LA = GenLA(GVCI)

module GenIV5 = GenGE(struct
  module Det =       AbstractDet
  module PivotF =    FullPivot
  module PivotRep =  PermList
  module Update =    FractionFreeUpdate
  module Input =     InpJustMatrix
  module Output =    OutDetRank end)
```

Design Concepts

Design Dim.	Abstracts
Domain	Matrix values
Normalization	domain needs it?
ZeroEquivalence	decidability of $= 0$
Representation	Matrix representation
Fraction-free	use of division
Pivoting Strategy	ex:use length?
Pivoting Choice	no/row/column/total
Pivot Rep	list, array, matrix
Full Division	division in domain
Rank	track rank?
Determinant	determinant tracking
Output	choice of output
Packed	L and U as one?
Lower	track lower L ?

Design Dim.	Abstracts
Code Rep	codegen options
UserInformation	user-feedback
Augmented	matrix is augmented
Input	choice of input
Logging	trace algorithm
Structure	ex: tri-diagonal
Warning	warn on 0? pivot
In-place	res. stored in input
Error-on-singular	input (near) singular
Conditioning	cond. numb. est.

Design space for LU Decomposition $\geqslant 25$ dimensional!

Little Theories

```
Theory CarrierType  carrier:type
Theory Carrier extends CarrierType  U:carrier
Theory PointedCarrier extends Carrier  e:U
Theory Unit extends Carrier    property singular(x) := forall y in U. x = y
Theory One extends Unit  axiom singular(e)
Theory BinaryOperation extends Carrier  op:(U, U)->U
Theory PointedBinaryOperation extends Carrier    combines BinaryOperation, PointedCarrier
Theory Two  combines One, One along Carrier
Theory Bool  using Two with e'1 = true, e'2 = false
...
Theory Cancellative extends Magma    combines LeftCancellative, RightCancellative
Theory Unital extends Magma  using Identity
Theory QuasiGroup extends Magma  using Cancellative
Theory Loop extends Magma  combines Unital, QuasiGroup
Theory SemiGroup extends Magma  using Associativity
Theory Band extends SemiGroup  using Idempotency
Theory Group extends Magma  combines Loop, Associativity
Theory AbelianGroup extends Group  using Commutativity
Theory Monoid extends Magma  combines Unital, SemiGroup
Theory CommutativeMonoid extends Monoid  using Commutativity
```