Symbolic Computation:

Self-application of Algorithmic Mathematics"

Bruno Buchberger
RISC (Research Institute for Symbolic Computation)
Johannes Kepler University, Linz, Austria

Invited Talk at MAP 2006 (Mathematics, Algorithms, Proofs)
Castro-Urdiales, Spain

A View on Mathematics

Example: Proving and Computing

Example: Proving by Reduction to Algebra

Example: Inventing by Schemes

Example: Inventing by Failing Proofs

A View on Mathematics

Example: Proving and Computing

Example: Proving by Reduction to Algebra

Example: Inventing by Schemes

Example: Inventing by Failing Proofs

The Simple Message

We are entering a new era of mathematics:

"doing" mathematics by applying algorithmic mathematics on the meta-level

invent definitions	discover and verify propositions	invent problems	invent and verify methods (algorithms)	apply algorithms

Self-application of (Algorithmic) Mathematics

■ Can / will / should revolutionize the way we do mathematics in 21st century

Mathematics:

- globally accessible *formal* (logic / computer based) knowledge bases
- expanded and verified by algorithmic (verified) *reasoners*

- *self*-expansion and expansion under *guidance*
- multiple "*views*"

Kind of "(anti-) bourbakism" of the 21st century.

- Not a Bourbakism of content but a Bourbakism of methodology.
- Not a Bourbakism that excludes "the computer" (i.e. algorithms) but a Bourbakism that puts the computer into the center of mathematics (both on the object level and the meta level).
- Not a Bourbakism that builds up *one* view of mathematics but gives us the tools for easily generating many views of mathematics.

The Time in History for Achieving this Aspiration

The ingredients are here

- new algorithms based on new and deep mathematical results (cad, PZ...theory, ...)
- deep understanding of logic
- marvelous software technology
- drastic improvement in hardware
- the web
- the mathematics and logic software systems (*Mathematica*, ..., Coq, ...)

Main Obstacle

- Our systems are not yet good (practical, comprehensive, uniform ...) enough for making them attractive for "working mathematicians".

Editorial of the J of Symbolic Computation, B. B. 1985

Symbolic Computation = "Computer Algebra" + "Computer Logic"

"object" level "meta" level

"object", "meta": relative notions!

Since then,

some progress has been made in the unification of the CA and CL communities.

However, not sufficiently much. In particular, not on the (logic and software) systems level!

Even some reverse tendencies.

Some positive signals: Calculemus Network, MKM Network, MAP.

Future Symbolic Systems

- Include general and special, interactive and fully automated, reasoners.
- Include hierarchically structured formal mathematical knowledge libraries. *)
- Have one language for mathematical knowledge and algorithms.
- Have the algorithms formally specified and verified.
- Have the algorithmic reasoners formally specified and verified.
- Include tools for managing large mathematical knowledge (and algorithm) libraries. (Store knowledge, retrieve knowledge, re-use knowledge, decide about originality of knowledge, re-organize knowledge, design "views" about mathematical areas, ...)

*) See the NIST project on Special Functions. However, mathematical knowledge is more than identities (inequalities, ...).

The Theorema Project

The Theorema project aims at prototyping such a system. There are a couple of other groups with the same aim (e.g. MIZAR, ...), see Calculemus and MKM network.

The *Theorema* group: B. B. (leader), T. Jebelean, T. Kutsia, F. Piroi, M. Rosenkranz, W. Windsteiger, and PhD students.

Some Reasoners in Theorema:

- Predicate logic: natural deduction, S-decomposition
- Elementary analysis: PCS (alternating quantifiers)
- Set theory
- Induction on natural numbers, on tuples
- Equality, sequence variables
- Combinatorial identities
- Geometry (based on algebraic methods like Gröbner bases)
- Algorithms for symbolic functional analysis (boundary value problems)
- "Lazy Thinking" method for lemma and algorithm invention
- Tools for structuring knowledge bases: functors, schemes, and others

A View on Mathematics

Example: Proving and Computing

Example: Proving by Reduction to Algebra

Example: Inventing by Schemes

Example: Inventing by Failing Proofs

The Point:

(Predicate) logic as both a logic language and a programming language.

Proving (algorithms) and computing (with these algorithms) in the same system.

Computing as a special case of proving.

Proving

```
Definition["addition", any[m, n],
    m + 0 = m          " +0:"
    m + n⁺ = (m + n)⁺  " + .:"]
```

```
Proposition["left zero", any[m, n],
    0 + n = n  "0+"]
```

```
Prove[Proposition["left zero"],
    using → ⟨Definition["addition"]⟩,
    by → NNEqIndProver,

    ProverOptions → {TermOrder → LeftToRight},
 transformBy → ProofSimplifier, TransformerOptions → {branches → {Proved}}];
```

Computing in the Same System

```
Compute[0⁺⁺ + 0⁺⁺⁺, using → ⟨Definition["addition"]⟩]
```

$((((0^{+})^{+})^{+})^{+})^{+}$

Another Example

Similarly, using our set theory prover, we could prove that

$$\begin{aligned}
&\mathbf{any}[\text{is-set}[A]]:\\
&\quad \mathcal{P}[A] = \{\{\}\} \Leftarrow (A = \{\})\\
&\quad \mathcal{P}[A] = \mathbf{where}\Big[a = \text{an-element}[A],\ P = \mathcal{P}[A \ominus \{a\}],\\
&\qquad P \cup \Big\{(\{a\} \cup B) \underset{B \in P}{|}\Big\}\Big]
\end{aligned}$$

Then we could use this knowledge and compute:

```
𝒫[{}]
```

{{}}

```
𝒫[{3}]
```

{{}, {3}}

```
𝒫[{1, 3}]
```

{{}, {1}, {3}, {1, 3}}

```
P[{1, 3, 4, 8}]
```

```
{{}, {1}, {3}, {4}, {8}, {1, 3}, {1, 4}, {1, 8}, {3, 4}, {3, 8},
 {4, 8}, {1, 3, 4}, {1, 3, 8}, {1, 4, 8}, {3, 4, 8}, {1, 3, 4, 8}}
```

A View on Mathematics

Example: Proving and Computing

Example: Proving by Reduction to Algebra

Example: Inventing by Schemes

Example: Inventing by Failing Proofs

The Point

Practical reasoning systems should have special reasoners for special theories.

Special reasoners often reduce proving in the special theory to "solving" in some algebraic domains.

In other words, reduction (by some simple logic steps) of algorithmic reasoning to (sophisticated) algorithmic algebra.

invent definitions	discover and verify propositions	invent problems	invent and verify methods (algorithms)	apply algorithms

The PCS Method for Analysis Proving (BB 2001)

This method reduces proving in elementary analysis (formulae with "alternating quantifiers" on functions) systematically to the solution of inequalities over the real numbers.

Produces "natural" proofs that also contain algorithmic information.

A Proof Generated by PCS

```
Definition["limit:", any[f, a],
    limit[f, a] ⟺ ∀ ∃ ∀ |f[n] − a| < ϵ]
                  ϵ  N  n
                 ϵ>0   n≥N
```

```
Proposition["limit of sum", any[f, a, g, b],
    (limit[f, a] ∧ limit[g, b])  ⇒  limit[f + g, a + b]]
```

```
Definition["+:", any[f, g, x],
    (f + g)[x] = f[x] + g[x]]
```

```
Lemma["|+|", any[x, y, a, b, δ, ϵ],
    (|(x + y) − (a + b)| < (δ + ϵ))  ⟸  (|x − a| < δ ∧ |y − b| < ϵ)]
```

```
Lemma["max", any[m, M1, M2],
    m ≥ max[M1, M2]  ⇒  (m ≥ M1 ∧ m ≥ M2)]
```

```
Theory["limit",
    Definition["limit:"]
    Definition["+:"]
    Lemma["|+|"]
    Lemma["max"]      ]
```

```
Prove[Proposition["limit of sum"], using → Theory["limit"], by → PCS]
```

- ProofObject -

The following proof is generated fully automatically by the PCS prover:

Prove:

(Proposition (limit of sum)) $\underset{f,a,g,b}{\forall}(\mathrm{limit}[f,a] \wedge \mathrm{limit}[g,b] \Rightarrow \mathrm{limit}[f+g,a+b])$,

under the assumptions:

(Definition (limit:)) $\underset{f,a}{\forall}\left(\mathrm{limit}[f,a] \Leftrightarrow \underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n\geq N}}{\forall}(|f[n]-a|<\epsilon)\right)$,

(Definition (+:)) $\underset{f,g,x}{\forall}((f+g)[x]=f[x]+g[x])$,

(Lemma (|+|)) $\underset{x,y,a,b,\delta,\epsilon}{\forall}(|(x+y)-(a+b)|<\delta+\epsilon \Leftarrow (|x-a|<\delta \wedge |y-b|<\epsilon))$,

(Lemma (max)) $\underset{m,M1,M2}{\forall}(m \geq \max[M1,M2] \Rightarrow m \geq M1 \wedge m \geq M2)$.

We assume

(1) $\mathrm{limit}[f_0,a_0] \wedge \mathrm{limit}[g_0,b_0]$,

and show

(2) $\mathrm{limit}[f_0+g_0,a_0+b_0]$.

Formula (1.1), by (Definition (limit:)), implies:

(3) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n\geq N}}{\forall}(|f_0[n]-a_0|<\epsilon)$.

By (3), we can take an appropriate Skolem function such that

(4) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{\substack{n \\ n\geq N_0[\epsilon]}}{\forall}(|f_0[n]-a_0|<\epsilon)$,

Formula (1.2), by (Definition (limit:)), implies:

(5) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n\geq N}}{\forall}(|g_0[n]-b_0|<\epsilon)$.

By (5), we can take an appropriate Skolem function such that

(6) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{\substack{n \\ n\geq N_1[\epsilon]}}{\forall}(|g_0[n]-b_0|<\epsilon)$,

Formula (2), using (Definition (limit:)), is implied by:

(7) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n\geq N}}{\forall}(|(f_0+g_0)[n]-(a_0+b_0)|<\epsilon)$.

We assume

(8) $\epsilon_0 > 0$,

and show

(9) $\underset{N}{\exists}\ \underset{\substack{n \\ n\geq N}}{\forall}\ (|(f_0 + g_0)[n] - (a_0 + b_0)| < \epsilon_0)$.

We have to find N_2^* such that

(10) $\underset{n}{\forall}\ (n \geq N_2^* \Rightarrow |(f_0 + g_0)[n] - (a_0 + b_0)| < \epsilon_0)$.

Formula (10), using (Definition (+:)), is implied by:

(11) $\underset{n}{\forall}\ (n \geq N_2^* \Rightarrow |(f_0[n] + g_0[n]) - (a_0 + b_0)| < \epsilon_0)$.

Formula (11), using (Lemma (|+|)), is implied by:

(12) $\underset{\substack{\delta,\epsilon \\ \delta+\epsilon=\epsilon_0}}{\exists}\ \underset{n}{\forall}\ (n \geq N_2^* \Rightarrow |f_0[n] - a_0| < \delta \wedge |g_0[n] - b_0| < \epsilon)$.

We have to find δ_0^*, ϵ_1^*, and N_2^* such that

(13) $(\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge \underset{n}{\forall}\ (n \geq N_2^* \Rightarrow |f_0[n] - a_0| < \delta_0^* \wedge |g_0[n] - b_0| < \epsilon_1^*)$.

Formula (13), using (6), is implied by:

$(\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge \underset{n}{\forall}\ (n \geq N_2^* \Rightarrow \epsilon_1^* > 0 \wedge n \geq N_1[\epsilon_1^*] \wedge |f_0[n] - a_0| < \delta_0^*)$,

which, using (4), is implied by:

$(\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge \underset{n}{\forall}\ (n \geq N_2^* \Rightarrow \delta_0^* > 0 \wedge \epsilon_1^* > 0 \wedge n \geq N_0[\delta_0^*] \wedge n \geq N_1[\epsilon_1^*])$,

which, using (Lemma (max)), is implied by:

(14) $(\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge \underset{n}{\forall}\ (n \geq N_2^* \Rightarrow \delta_0^* > 0 \wedge \epsilon_1^* > 0 \wedge n \geq \max[N_0[\delta_0^*], N_1[\epsilon_1^*]])$.

Formula (14) is implied by

(15) $(\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge \delta_0^* > 0 \bigwedge \epsilon_1^* > 0 \bigwedge \underset{n}{\forall}\ (n \geq N_2^* \Rightarrow n \geq \max[N_0[\delta_0^*], N_1[\epsilon_1^*]])$.

Partially solving it, formula (15) is implied by

(16) $(\delta_0^* + \epsilon_1^* = \epsilon_0) \wedge \delta_0^* > 0 \wedge \epsilon_1^* > 0 \wedge (N_2^* = \max[N_0[\delta_0^*], N_1[\epsilon_1^*]])$.

Now,

$(\delta_0^* + \epsilon_1^* = \epsilon_0) \wedge \delta_0^* > 0 \wedge \epsilon_1^* > 0$

can be solved for δ_0^* and ϵ_1^* by a call to Collins cad–method yielding a sample solution

$\delta_0^* \leftarrow \frac{\epsilon_0}{2}$,

$\epsilon_1^* \leftarrow \frac{\epsilon_0}{2}$.

Furthermore, we can immediately solve

$N_2^* = \max[N_0[\delta_0^*], N_1[\epsilon_1^*]]$

for N_2^* by taking

$N_2^* \leftarrow \max[N_0[\frac{\epsilon_0}{2}], N_1[\frac{\epsilon_0}{2}]]$.

Hence formula (16) is solved, and we are done.

□

A View on Mathematics

Example: Proving and Computing

Example: Proving by Reduction to Algebra

Example: Inventing by Schemes

Example: Inventing by Failing Proofs

The Point

(Some part of) invention in mathematics may be mimicked by the application (instantiation) of "schemes".

In other words, schemes are an abstract formulation of accumulated "mathematical experience".

Schemes can be used for inventing definitions, propositions, problems, and methods (algorithms).

invent definitions	discover and verify propositions	invent problems	invent and verify methods (algorithms)	apply algorithms

Trivial example

A "typical" formula:

$$\underset{a,b,f,g}{\forall} (\mathrm{P}[f, a] \wedge \mathrm{P}[g, b]) \Rightarrow \mathrm{P}[\mathrm{F}[f, g], \mathrm{G}[a, b]]$$

Can be used as a "scheme":

$$\underset{\mathrm{P},\mathrm{F},\mathrm{G}}{\forall} \left(\mathrm{Monotony}[\mathrm{P}, \mathrm{F}, \mathrm{G}] \Leftrightarrow \underset{a,b,f,g}{\forall} (\mathrm{P}[f, a] \wedge \mathrm{P}[g, b]) \Rightarrow \mathrm{P}[\mathrm{F}[f, g], \mathrm{G}[a, b]]\right)$$

Given a knowledge base in which 'Limit', '+', and '+' occurs, we can apply the above scheme for "inventing" (proposing, conjecturing) a proposition:

$$\mathrm{Monotony}[\mathrm{Limit}, +, +]$$

i.e.

$$\underset{a,b,f,g}{\forall} (\mathrm{Limit}[f, a] \wedge \mathrm{Limit}[g, b]) \Rightarrow \mathrm{Limit}[f + g, a + b]$$

'Monotony' is a "relator" or (the description of) a "category".

Another Example

A formula (in the special theory of tuples):

$$\underset{x}{\forall}\left(\mathbf{N}[x] = \begin{cases} \mathbf{S}[x] & \Leftarrow \text{is-trivial-tuple}[x] \\ \mathbf{M}[\text{sorted}[\mathbf{L}[x]], \text{sorted}[\mathbf{R}[x]]] & \Leftarrow \text{otherwise} \end{cases}\right)$$

A scheme:

$$\underset{N,S,M,L,R}{\forall} \text{Divide-and-Conquer}[\mathbf{N}, \mathbf{S}, \mathbf{M}, \mathbf{L}, \mathbf{R}] \Leftrightarrow$$
$$\underset{x}{\forall}\left(\mathbf{N}[x] = \begin{cases} \mathbf{S}[x] & \Leftarrow \text{is-trivial-tuple}[x] \\ \mathbf{M}[\text{sorted}[\mathbf{L}[x]], \text{sorted}[\mathbf{R}[x]]] & \Leftarrow \text{otherwise} \end{cases}\right)$$

Given a knowledge base in which 'identity', 'merge', 'left' and 'right' occurs, we can apply the above scheme for "inventing" (proposing, conjecturing) a sorting *algorithm*. It could also be applied, in a different context, for *defining* a new notion N in terms of known notions S, M, L, R.

'Divide–and–Conquer' is a "relator" with some functional flavor. It is a "functor". (In Theorema, a special notation is available for functors.)

This functor captures an essential mathematical idea for invention notions and solutions to problems.

Another Example

A formula:

$$\underset{F}{\forall} \mathbf{A}[F] = \mathbf{A}[F, \text{pairs}[F]]$$

$$\underset{F}{\forall} \mathbf{A}[F, \langle\rangle] = F$$

$$\underset{F,g1,g2,\bar{p}}{\forall} \mathbf{A}[F, \langle\langle g1, g2\rangle, \bar{p}\rangle] =$$
$$\text{where}\Big[f = \mathbf{lc}[g1, g2], h1 = \text{trd}[\text{rd}[f, g1], F], h2 = \text{trd}[\text{rd}[f, g2], F],$$
$$\begin{cases} \mathbf{A}[F, \langle\bar{p}\rangle] & \Leftarrow h1 = h2 \\ \mathbf{A}\Big[F \smallfrown \mathbf{df}[h1, h2], \langle\bar{p}\rangle \asymp \Big\langle \underset{k=1,\ldots,|F|}{\big|} \langle F_k, \mathbf{df}[h1, h2]\rangle \Big\rangle\Big] & \Leftarrow \text{otherwise} \end{cases}\Big]$$

A scheme (a "functor"):

$$\underset{A,lc,df}{\forall} \mathbf{pair\text{-}completion}[A, lc, df] \Leftrightarrow$$

$$\underset{F}{\forall} A[F] = A[F, \mathrm{pairs}[F]]$$

$$\underset{F}{\forall} A[F, \langle\rangle] = F$$

$$\underset{F,g1,g2,\bar{p}}{\forall} A[F, \langle\langle g1, g2\rangle, \bar{p}\rangle] = \; ....lc \;...\; df \;...$$

Functor Notation in Theorema

```
Definition["Groebner extension", any[R],
 Groebner-extension[R] (* the Groebner extension of a reduction ring R *) =
  Functor[N, any[C, k, p, p̄, q, q̄, x, X, y, ȳ, Y],
```

```
rd_N[x, y] = x -_R rdm_R[x, y] *_R y

trd_N[x, Y] = trd_N[x, Y, 1]
trd_N[x, Y, k] =
  { x                                          ⇐ k > |Y|
  { where[x1 = rd_N[x, Y_k],                   ⇐ otherwise
  {   { trd_N[x1, Y, 1]      ⇐ x >_R x1
  {   { trd_N[x, Y, k + 1]   ⇐ otherwise   ]

cpd_N[x, y] (* the critical pair difference of x and y *) =
 where[lxy = lcrd_R[x, y], rd_N[lxy, x] -_R rd_N[lxy, y]]

Gb_N[X] = Gb_N[X, pairs[X]]
Gb_N[X, 〈〉] = X
Gb_N[X, 〈〈x, y〉, p̄〉] =
 where[h = trd_N[cpd_N[x, y], X],
  { Gb_N[X, 〈p̄〉]                                        ⇐ h = 0_N
  { Gb_N[X ⌢ h, 〈p̄〉 ≍ 〈〈X_k, h〉 | k=1,…,|X|〉]          ⇐ otherwise   ]

rdGb_N[X] (* a reduced Groebner basis of 〈p,p̄〉 *) = ard_N[Gb_N[X]]
ard_N[〈〉] = 〈〉
ard_N[〈p, q̄〉] = ard_N[〈〉, p, 〈q̄〉]
ard_N[X, p, 〈〉] = where[h = trd_N[p, X],
  { X      ⇐ h = 0_N
  { X ⌢ p  ⇐ otherwise   ]
ard_N[X, p, 〈q, q̄〉] = where[h = trd_N[p, X ≍ 〈q, q̄〉],
  { ard_N[X, q, 〈q̄〉]       ⇐ h = 0_N
  { ard_N[X ⌢ p, q, 〈q̄〉]   ⇐ otherwise   ]

tcrd_N[x, Y] (* the total cofactor reduction of x modulo tuple Y *) = tcrd_N[x, Y,

tcrd_N[x, Y, k, C] (* the total cofactor reduction of x modulo the k-th element

  { 〈x, C〉                                                ⇐ k > |Y|
  { where[c = rdm_N[x, Y_k], x1 = x -_N c *_N Y_k,         ⇐ otherwise
  {   { tcrd_N[x1, Y, 1, C_{k←C_k +_N c}]   ⇐ x >_N x1
  {   { tcrd_N[x, Y, k + 1, C]              ⇐ otherwise   ]
```

Another Example

The scheme

$$\underset{F,C,D}{\forall}\ \text{conservation-theorem}[F, C, D] \Longleftrightarrow \underset{f}{\forall}\ (C[f] \Rightarrow D[F[f]])$$

captures the ubiquitous idea of a "conservation" theorem: If the domain f is in the category C then F[f] (the domain that results by applying functor F to f) is in category D.

Summarizing

Given a knowledge base on certain functions and predicates (of any order), one may apply (instantiate) schemes for generating ("inventing") lots of proposals for (interesting) definitions, propositions, problems, and methods (algorithms) for solving problems.

This may semi-automate the "easy" part of exploring a theory in a systematic way and may take away tedious formula typing.

Then we call semi-automated provers for disproving / disproving part of the proposed knowledge. This may take away another part of tedious exploration.

Whatever is left, is (called) "non-trivial". (This is a relative notion of "triviality" !)

This approach is a "bottom-up" approach. Now let's put in some salt by a "top-down" idea: learning from failing proofs and thereby invent something.

We illustrate the idea in the case of algorithm invention ("algorithm synthesis").

A View on Mathematics

Example: Proving and Computing

Example: Proving in Elementary Analysis

Example: Inventing by Schemes

Example: Algorithm Inventing by Failing Proofs

The Algorithm Invention ("Synthesis") Problem

Given a problem specification P (in predicate logic), find an algorithm A such that

$$\underset{x}{\forall} \mathbf{P}[x, A[x]].$$

Examples of specifications P:

```
P[x, y] ⇔ is-greater[x, y]
P[x, y] ⇔ is-sorted-version[x, y]
P[x, y] ⇔ has-derivative[x, y]
P[x, y] ⇔ are-factors-of[x, y]
P[x, y] ⇔ is-Gröbner-basis[x, y]
 ....
```

A general algorithm S for "all" P cannot exist but ...

Algorithm Synthesis by "Lazy Thinking" (BB 2002)

"Lazy Thinking" Method for Algorithm Synthesis =

My Advice to "Humans" (or "Computers") How to Invent Algorithms.

Given: A problem P. Find: An algorithm A for P.

- Learn how to prove.
- Completely understand the problem P. ("Specification" of the problem.)
- Collect (discover, prove) "complete" knowledge on the auxiliary notion appearing in the problem P.
- Consider known fundamental ideas of how to structure algorithms in terms of subalgorithms ("algorithm schemes A").

  Try one scheme A after the other.
- For the chosen scheme A, try to prove $\underset{x}{\forall}$ P[x, A[x]]: From the failing proof construct specifications for the subalgorithms B occurring in A.

Literature

There is a rich literature on algorithm synthesis methods, see survey

[Basin et al. 2004] D. Basin, Y. Deville, P. Flener, A. Hamfelt, J. F. Nilsson. Synthesis of Programs in Computational Logic. In: M. Bruynooghe, K. K. Lau (eds.), Program Development in Computational Logic, Lecture Notes in Computer Science, Vol. 3049, Springer, 2004, pp. 30-65.

My method is in the class of "scheme-based" methods. Closest (but essentially different):

[Lau et al. 1999] K. K. Lau, M. Ornaghi, S. Tärnlund. Steadfast logic programs. Journal of Logic Programming, 38/3, 1999, pp. 259-294.

And the work of A. Bundy and his group (U of Edinburgh) on the automated invention of induction schemes.

Example: Synthesis of Merge-Sort [BB et al. 2003]

Problem: Synthesize "sorted" such that

$$\underset{x}{\forall}\ \textbf{is-sorted-version}[x, \textbf{sorted}[x]].$$

("Correctness Theorem")

Knowledge on Problem:

$$\underset{x,y}{\forall}\left(\textbf{is-sorted-version}[x, y] \Leftrightarrow \begin{array}{l}\textbf{is-sorted}[y]\\ \textbf{is-permuted-version}[x, y]\end{array}\right)$$

$$\textbf{is-sorted}[\langle\rangle]$$

$$\underset{x}{\forall}\ \textbf{is-sorted}[\langle x\rangle]$$

$$\underset{x,y,z}{\forall}\left(\textbf{is-sorted}[\langle x, y, z\rangle] \Leftrightarrow \begin{array}{l}x \geq y\\ \textbf{is-sorted}[\langle y, z\rangle]\end{array}\right)$$

etc.

An Algorithm Scheme: Divide and Conquer

$$\underset{x}{\forall}\left(\textbf{sorted}[x] = \begin{cases}\mathbf{S}[x] & \Leftarrow \textbf{is-trivial-tuple}[x]\\ \mathbf{M}[\textbf{sorted}[\mathbf{L}[x]], \textbf{sorted}[\mathbf{R}[x]]] & \Leftarrow \textbf{otherwise}\end{cases}\right)$$

S, M, L, R are unknowns.

We now start an (automated) induction prover for proving the correctness theorem and analyze the failing proof: see notebooks with failing proofs.

Automated Invention of Sufficient Specifications for the Subalgorithms

A simple (but amazingly powerful) rule (m ... an unknown subalgorithm):

Collect temporary assumptions T[x0, ... A [ ], ...]

and temporary goals G[x0, ...m [A [ ]]]

and produces specification

$$\underset{X, \ldots, Y, \ldots}{\forall} \left(T[X, \ldots Y, \ldots] \Rightarrow G[Y, \ldots m[Y]]\right).$$

Details: see papers [BB 2003] and example.

The Result of Applying Lazy Thinking in the Sorting Example

Lazy Thinking, automatically (in approx. 2 minutes on a laptop using the *Theorema* system), finds the following specifications for the sub-algorithms that provenly guarantee the correctness of the above algorithm (scheme):

$$\underset{x}{\forall} \left(\text{is-trivial-tuple}[x] \Rightarrow \mathbf{S}[x] = x\right)$$

$$\underset{y,z}{\forall} \left(\begin{matrix} \text{is-sorted}[y] \\ \text{is-sorted}[z] \end{matrix} \Rightarrow \begin{matrix} \text{is-sorted}[\mathbf{M}[y, z]] \\ \mathbf{M}[y, z] \approx (y \asymp z) \end{matrix} \right)$$

$$\underset{x}{\forall} \left(\mathbf{L}[x] \asymp \mathbf{R}[x] \approx x\right)$$

Note: the specifications generated are not only sufficient but natural !

What Do We Have Now?

- **Case A:** We find algorithms S0, M0, L0, R0 in our knowledge base for which the properties specified above for S, M, L, R are already contained in the knowledge base or can be derived (proved) from the knowledge base.

  In this case, we are done, i.e. we have synthesized a sorting algorithm.

- **Case B:** We do not find such algorithms S0, M0, L0, R0 in our knowledge base.

  In this case, we apply Lazy Thinking again in order to synthesize appropriate S, M, L, R

  until we arrive at sub-sub-...-algorithms in our knowledge base (e.g. the basic operations of tuple theory like append, prepend etc.)

Case B can be avoided, if we proceed systematically bottom-up ("complete theory exploration" in layers).

Example: Synthesis of Insertion-Sort

Synthesize A such that

$$\underset{x}{\forall}\ \text{is-sorted-version}[x, A[x]].$$

Algorithm Scheme: "simple recursion"

$$A[\langle\rangle] = \mathbf{c}$$
$$\underset{x}{\forall}\ A[\langle x\rangle] = \mathbf{s}[\langle x\rangle]$$
$$\underset{x,\bar{y}}{\forall}\ (A[\langle x, \bar{y}\rangle] = \mathbf{i}[x, A[\langle\bar{y}\rangle]])$$

Lazy Thinking, automatically (in approx. 2 minutes on a laptop using the *Theorema* system), finds the following specifications for the auxiliary functions

$$\mathbf{c} = \langle\rangle$$
$$\underset{x}{\forall}\ (\mathbf{s}[\langle x\rangle] = \langle x\rangle)$$
$$\underset{x,\bar{y}}{\forall}\ \left(\text{is-sorted}[\langle\bar{y}\rangle] \Rightarrow \begin{array}{l}\text{is-sorted}[\mathbf{i}[x, \langle\bar{y}\rangle]] \\ \mathbf{i}[\langle x, \bar{y}\rangle] \approx (x \smile \langle\bar{y}\rangle)\end{array}\right)$$

How Far Can We Go With the Method ?

Can we automatically synthesize algorithms for non-trivial problems? What is "non-trivial"?

Example of a non-trivial problem (?): construction of Gröbner bases.

"Non-trivial": The invention of the notion of S-polynomial and the characterization of Gröbner-bases by finitely many S-polynomial checks.

With the "Lazy Thinking" method, it is possible to invent the essential idea of the B.B.'s Gröbner bases algorithm fully automatically: See [BB 2005].

The Problem of Constructing Gröbner Bases

Find algorithm Gb such that

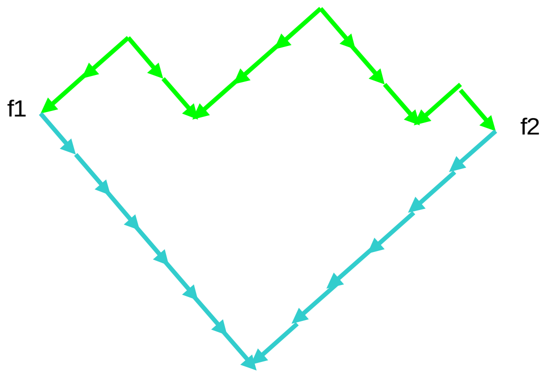
$$\underset{\text{is-finite}[F]}{\forall} \begin{pmatrix} \text{is-finite}[\text{Gb}[F]] \\ \text{is-Gröbner-basis}[\text{Gb}[F]] \\ \text{ideal}[F] = \text{ideal}[\text{Gb}[F]]. \end{pmatrix}$$

$$\text{is-Gröbner-basis}[G] \Leftrightarrow \text{is-confluent}[\rightarrow_G].$$

$\rightarrow_G$ ... a division step.

Confluence of Division $\rightarrow_G$

$$\text{is-confluent}[\rightarrow] :\Leftrightarrow \underset{f1,f2}{\forall} (f1 \leftrightarrow^* f2 \Rightarrow f1 \downarrow^* f2)$$



Knowledge on the Concepts Involved

$$\mathrm{h1} \rightarrow_G \mathrm{h2} \;\Rightarrow\; \mathrm{p}\,.\,\mathrm{h1} \rightarrow_G \mathrm{p}\,.\,\mathrm{h2}$$

etc.

Algorithm Scheme "Critical Pair / Completion"

```
A[F] = A[F, pairs[F]]
A[F, ⟨⟩] = F

A[F, ⟨⟨g1, g2⟩, p̄⟩] =
 where[f = lc[g1, g2], h1 = trd[rd[f, g1], F], h2 = trd[rd[f, g2], F],
  { A[F, ⟨p̄⟩]                                                   ⇐ h1 = h2
  { A[F ⌢ df[h1, h2], ⟨p̄⟩ ≍ ⟨⟨F_k, df[h1, h2]⟩ | k=1,…,|F| ⟩]   ⇐ otherwise   ]
```

This scheme can be tried in any domain, in which we have a reduction operation rd that depends on sets F of objects and a Noetherian relation $\succ$ which interacts with rd in the following natural way:

$$\underset{f,g}{\forall}\,(f \succeq \mathrm{rd}[f, g]).$$

The Essential Problem

The problem of synthesizing a Gröbner bases algorithm can now be also stated by asking whether starting with the proof of

$$\underset{F}{\forall}\begin{pmatrix}\text{is-finite}[\,A[F]\,]\\ \text{is-Gröbner-basis}[\,A[F]]\\ \text{ideal}[F] = \text{ideal}[\,A[F]].\end{pmatrix}$$

using the above scheme for A we can *automatically* produce the idea that

```
lc[g1, g2] = lcm[lp[g1], lp[g2]]
```

and

```
df[h1, h2] = h1 - h2
```

and prove that the idea is correct.

Now Start the (Automated) Correctness Proof

With current theorem proving technology, in the *Theorema* system (and other provers), the proof attempt can be done automatically. (Ongoing PhD thesis by A. Craciun.)

Details

It should be clear that, if the algorithm terminates, the final result is a finite set (of polynomials) G that has the property

$$\underset{g1,g2\in G}{\forall}\left(\text{where}\left[f = \text{lc}[g1, g2],\ h1 = \text{trd}[\text{rd}[f, g1], F],\ h2 = \text{trd}[\text{rd}[f, g2], F],\ \bigvee\begin{cases} h1 = h2 \\ \text{df}[h1, h2] \in G \end{cases}\right]\right).$$

We now try to prove that, if G has this property, then

```
is-finite[G],
ideal[F] = ideal[G],
is-Gröbner-basis[G],
   i.e. is-Church-Rosser[ →G ].
```

Here, we only deal with the third, most important, property.

Using Available Knowledge

Using Newman's lemma and some elementary properties it can be shown that it is sufficient to prove

$$\text{is-Church-Rosser}[\rightarrow_G] \Leftrightarrow \underset{p}{\forall}\ \underset{f1,f2}{\forall}\left(\left(\begin{cases} p \rightarrow f1 \\ p \rightarrow f2 \end{cases}\right) \Rightarrow f1 \downarrow^* f2\right).$$

Newman's lemma (1942):

$$\text{is-Church-Rosser}[\rightarrow] \Leftrightarrow \underset{f,f1,f2}{\forall}\left(\left(\begin{cases} f \rightarrow f1 \\ f \rightarrow f2 \end{cases}\right) \Rightarrow f1 \downarrow^* f2\right).$$

Definition of "f1 and f2 have a common successor":

$$f1 \downarrow^* f2 \Leftrightarrow \underset{g}{\exists}\left(\begin{cases} f1 \rightarrow^* g \\ f2 \rightarrow^* g \end{cases}\right)$$

The (Automated) Proof Attempt

Let now the power product p and the polynomials f1, f2 be arbitary but fixed and assume

$$\begin{cases} p \rightarrow_G f1 \\ p \rightarrow_G f2. \end{cases}$$

We have to find a polyonomial g such that

$$f1 \rightarrow_G^* g,$$
$$f2 \rightarrow_G^* g.$$

From the assumption we know that there exist polynomials g1 and g2 in G such that

$$\begin{aligned} &\mathrm{lp}[g1] \mid p, \\ &f1 = \mathrm{rd}[p, g1], \\ &\mathrm{lp}[g2] \mid p, \\ &f2 = \mathrm{rd}[p, g2]. \end{aligned}$$

From the final situation in the algorithm scheme we know that for these g1 and g2

$$\bigvee \begin{cases} h1 = h2 \\ \mathrm{df}[h1, h2] \in G, \end{cases}$$

where

$$\begin{aligned} &h1 := \mathrm{trd}[f1', G],\ f1' := \mathrm{rd}[\mathrm{lc}[g1, g2], g1], \\ &h2 := \mathrm{trd}[f2', G],\ f2' := \mathrm{rd}[\mathrm{lc}[g1, g2], g2]. \end{aligned}$$

Case h1=h2

```
lc[g1, g2] →g1 rd[lc[g1, g2], g1] →G* trd[rd[lc[g1, g2], g1], G] =
 trd[rd[lc[g1, g2], g2], G] ←G* rd[lc[g1, g2], g2] ←g2 lc[g1, g2].
```

(Note that here we used the requirements rd[lc[g1,g2],g1]<lc[g1,g2] and rd[lc[g1,g2],g2]<lc[g1,g2].)

Hence, by elementary properties of polynomial reduction,

$$\underset{a,q}{\forall}\ (a\, q\, \mathtt{lc[g1, g2]} \rightarrow_{g1} a\, q\, \mathtt{rd[lc[g1, g2], g1]} \rightarrow_G^* a\, q\, \mathtt{trd[rd[lc[g1, g2], g1], G]} = a\, q\, \mathtt{trd[rd[lc[g1, g2], g2], G]} \leftarrow_G^* a\, q\, \mathtt{rd[lc[g1, g2], g2]} \leftarrow_{g2} a\, q\, \mathtt{lc[g1, g2]}).$$

Now we are stuck in the proof.

Now Use the Specification Generation Algorithm

Using the above specification generation rule, we see that we could proceed successfully with the proof if lc[g1,g2] satisfied the following requirement

$$\underset{p,g1,g2}{\forall}\left(\left(\left\{\begin{array}{l}\mathtt{lp[g1]} \mid p\\ \mathtt{lp[g2]} \mid p\end{array}\right.\right) \Rightarrow \left(\underset{a,q}{\exists}\ (p = a\, q\, \mathtt{lc[g1, g2]})\right)\right), \quad (\mathtt{lc\ requirement})$$

With such an lc, we then would have

```
p →g1 rd[p, g1] = a q rd[lc[g1, g2], g1] →G* a q trd[rd[lc[g1, g2], g1], G] =
   a q trd[rd[lc[g1, g2], g2], G] ←G* a q rd[lc[g1, g2], g2] = rd[p, g2] ←g2 p
```

and, hence,

```
f1 →G* a q trd[rd[lc[g1, g2], g1], G],
```

```
f2 →G* a q trd[rd[lc[g1, g2], g1], G],
```

i.e. we would have found a suitable g.

Summarize the (Automatically Generated) Specifications of the Subalgorithm lc

Using the above specification generation rule, we see that we could proceed successfully with the proof if lc[g1,g2] satisfied the following requirement

$$\underset{p,g1,g2}{\forall}\left(\left(\begin{cases} \mathrm{lp}[g1] \mid p \\ \mathrm{lp}[g2] \mid p \end{cases}\right) \Rightarrow (\mathrm{lc}[g1, g2] \mid p)\right),$$

and the requirements:

$$\mathrm{lp}[g1] \mid \mathrm{lc}[g1, g2],$$
$$\mathrm{lp}[g2] \mid \mathrm{lc}[g1, g2].$$

Now this problem can be attacked independently of any Gröbner bases theory, ideal theory etc.

A Suitable lc

$$\mathrm{lcp}[g1, g2] = \mathrm{lcm}[\mathrm{lp}[g1], \mathrm{lp}[g2]]$$

is a suitable function that satisfies the above requirements.

Eureka! The crucial function lc (the "critical pair" function) in the critical pair / completion algorithm scheme has been synthesized automatically!

Case h1≠h2

In this case, df[h1,h2]∈G:

In this part of the proof we are basically stuck right at the beginning.

We can try to reduce this case to the first case, which would generate the following requirement

$$\underset{h1,h2}{\forall}\left(h1 \downarrow^{*}_{\{\mathrm{df}[h1,h2]\}} h2\right) \quad (\mathrm{df\ requirement}).$$

Looking to the Knowledge Base for a Suitable df

(Looking to the knowledge base of elementary properties of polynomial reduction, it is now easy to find a function df that satifies (df requirement), namely

```
df[h1, h2] = h1 - h2,
```

because, in fact,

$$\underset{f,g}{\forall} (f \downarrow_{\{f-g\}}{}^{*} g).$$

Eureka! The function df (the "completion" function) in the critical pair / completion algorithm scheme has been "automatically" synthesized!)

Conclusion

I think we should put all our current achievements in "symbolic computatoin" (computer algebra, automated reasoning, etc.) together for coming up with coherent systems that allow to build up

- verified

- well-structured, restructurable, extensible

- globally accessible

- mathematical knowledge (definitions, propositions, problems, methods) bases.

Algorithms on the object level must be "shiftable" to the meta-level for becoming (part of) reasoning methods on the meta-level. (Reflexion is an essential ingredient of intelligent mathematical theory exploration.)

This will have a drastic effect on

- how we will be able to do research in mathematics
- how we will be able to store, publish, evaluate, organize, access mathematical knowledge
- how we will be able to teach mathematics.

References

On Gröbner Bases

[Buchberger 1970]

B. Buchberger. Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems (An Algorithmical Criterion for the Solvability of Algebraic Systems of Equations). Aequationes mathematicae 4/3, 1970, pp. 374-383. (English translation in: [Buchberger, Winkler 1998], pp. 535 -545.) Published version of the PhD Thesis of B. Buchberger, University of Innsbruck, Austria, 1965.

[Buchberger 1998]

B. Buchberger. Introduction to Gröbner Bases. In: [Buchberger, Winkler 1998], pp.3-31.

[Buchberger, Winkler, 1998]

B. Buchberger, F. Winkler (eds.). Gröbner Bases and Applications, Proceedings of the International Conference "33 Years of Gröbner Bases", 1998, RISC, Austria, London Mathematical Society Lecture Note Series, Vol. 251, Cambridge University Press, 1998.

[Becker, Weispfenning 1993]

T. Becker, V. Weispfenning. Gröbner Bases: A Computational Approach to Commutative Algebra, Springer, New York, 1993.

On Mathematical Knowledge Management

B. Buchberger, G. Gonnet, M. Hazewinkel (eds.)

Mathematical Knowledge Management.

Special Issue of Annals of Mathematics and Artificial Intelligence, Vol. 38, No. 1-3, May 2003, Kluwer Academic Publisher, 232 pages.

A.Asperti, B. Buchberger, J.H.Davenport (eds.)

Mathematical Knowledge Management.

Proceedings of the Second International Conference on Mathematical Knowledge Management (MKM 2003), Bertinoro, Italy, Feb.16-18, 2003, Lecture Notes in Computer Science, Vol. 2594, Springer, Berlin-Heidelberg-NewYork, 2003, 223 pages.

A.Asperti, G.Bancerek, A.Trybulec (eds.).

Proceedings of the Third International Conference on Mathematical Knowledge Management, MKM 2004,

Bialowieza, Poland, September 19-21, 2004, Lecture Notes in Computer Science, Vol. 3119, Springer, Berlin-Heidelberg-NewYork, 2004

■ On Theorema

[Buchberger et al. 2000]

B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, W. Windsteiger. The Theorema Project: A Progress Report. In: M. Kerber and M. Kohlhase (eds.), Symbolic Computation and Automated Reasoning (Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, August 6-7, 2000, St. Andrews, Scotland), A.K. Peters, Natick, Massachusetts, ISBN 1-56881-145-4, pp. 98-113.

■ On Theory Exploration and Algorithm Synthesis

[Buchberger 2000]

B. Buchberger. Theory Exploration with *Theorema*.

Analele Universitatii Din Timisoara, Ser. Matematica-Informatica, Vol. XXXVIII, Fasc.2, 2000, (Proceedings of SYNASC 2000, 2nd International Workshop on Symbolic and Numeric Algorithms in Scientific Computing, Oct. 4-6, 2000, Timisoara, Rumania, T. Jebelean, V. Negru, A. Popovici eds.), ISSN 1124-970X, pp. 9-32.

[Buchberger 2003]

B. Buchberger. Algorithm Invention and Verification by Lazy Thinking.

In: D. Petcu, V. Negru, D. Zaharie, T. Jebelean (eds), Proceedings of SYNASC 2003 (Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, October 1–4, 2003), Mirton Publishing, ISBN 973–661–104–3, pp. 2–26.

[Buchberger, Craciun 2003]

B. Buchberger, A. Craciun. Algorithm Synthesis by Lazy Thinking: Examples and Implementation in Theorema. in: Fairouz Kamareddine (ed.), Proc. of the Mathematical Knowledge Management Workshop, Edinburgh, Nov. 25, 2003, Electronic Notes on Theoretical Computer Science, volume dedicated to the MKM 03 Symposium, Elsevier, ISBN 044451290X, to appear.

[Buchberger 2005]

B. Buchberger.

Towards the Automated Synthesis of a Gröbner Bases Algorithm.

RACSAM (Review of the Royal Spanish Academy of Science), Vol. 98/1, 2005, pp. 65-75.