

# Automated Mathematical Theory Exploration: How Far Can We Go?

Bruno Buchberger

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

Seminar "The Role of Computer Science in Science"  
DERI, Innsbruck, December 14-15, 2006

The Mathematical Theory Exploration Process

Automated Mathematical Theory Exploration

Example: Groebner Bases Algorithm Synthesis

Conclusion

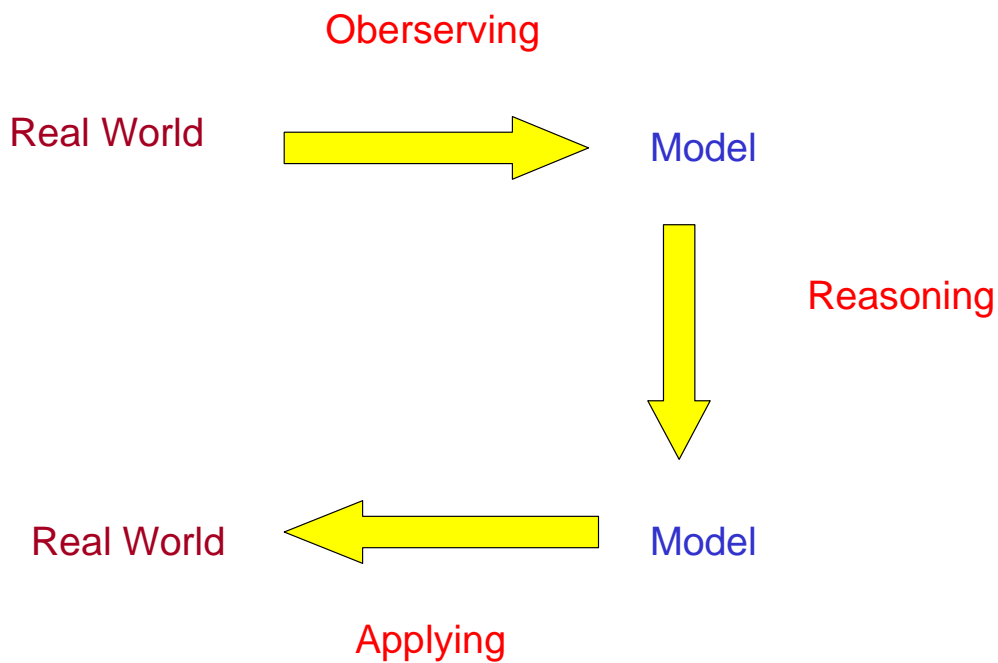
The Mathematical Theory Exploration Process

Automated Mathematical Theory Exploration

Example: Groebner Bases Algorithm Synthesis

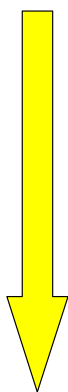
Conclusion

## The Scientific Approach



## Mathematical Theory Exploration

Model

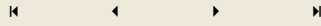


Model

- Invent Notions (Definitions, Axioms)
- Invent and Prove Theorems
- Invent Problems
- Invent and Prove Methods (Algorithms)

A mathematical theory: axioms, definitions, theorems, problem specification, methods.

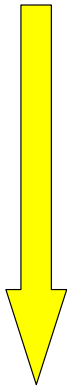
Mathematical theory exploration: the purest "intellectual" activity.



6 of 49

## Automated Mathematical Theory Exploration

Model



- Invent Notions (Definitions, Axioms)
- Invent and Prove Theorems
- Invent Problems
- Invent and Prove Methods (Algorithms)

Model

Automatic invention / proof of definitions, theorems, problems, methods.



7 of 49

## We Need a (Universal) Language for Expressing Mathematical Theories

Predicate logic: Example of a formula

$$\forall_{f,a} \text{limit}[f, a] \Leftrightarrow \forall_{\epsilon > 0} \exists N \forall_{n \geq N} |f[n] - a| < \epsilon$$

$$\text{limit}[f, a] \Leftrightarrow (\epsilon > 0 \Rightarrow (n \geq N \Rightarrow |-a + f[n]| < \epsilon))$$

Language constructs:

- constants for objects, functions, predicates,
- variables,

- propositional connectives like ' $\iff$ ',
- quantifiers like ' $\forall$ ', ' $\exists$ '.

Higher order: variables for functions and predicates.

## Examples are Taken from the Theorema Project

The Theorema project aims at prototyping features of a system for mathematical theory exploration.

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

The Theorema Language is a version of (higher order) predicate logic.

Some Automated Reasoners in Theorema:

- Predicate logic: natural deduction, S-decomposition
- Elementary analysis: PCS (alternating quantifiers)
- Set theory
- Induction on natural numbers, on tuples
- Equational logic with 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.

Computation within logic.

Two-dimensional user-definable syntax including "logicographic symbols".

Readable Proofs.

Meta-language: *Mathematica*.

Object language: higher-order predicate logic with sequence variables.

## Example of Some Exploration Steps: The Notion of Limit

Let's start from the situation:

- we already have explored the theory of the reals with  $+$ ,  $<$  ... "completely"
- we already have explored the operations  $\oplus$ , ... on sequences of reals

⏪ ⏩ ⏴ ⏵

11 of 49

## Introduction of the Notion of Limit by a Definition

```
Definition["limit:", any[f, a],
  limit[f, a]  $\Leftrightarrow$   $\forall_{\epsilon > 0} \exists_N \forall_{n \geq N} |f[n] - a| < \epsilon$ 
```

⏪ ⏩ ⏴ ⏵

12 of 49

## A Typical Theorem (How Do we Invent this Theorem?)

```
Proposition["limit of sum", any[f, a, g, b],
  (limit[f, a]  $\wedge$  limit[g, b])  $\Rightarrow$  limit[f  $\oplus$  g, a + b]]
```

⏪ ⏩ ⏴ ⏵

13 of 49

## The PCS Method for Automated Proving in Analysis (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.

Instead of a detailed explanation of the proof method, let's look to the proof generated for the above example of a proposition:

## ■ Initialize Theorema

⏪ ⏩ ⏴ ⏵

14 of 49

### (Automated) Proof of the Limit Theorema

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

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

Formulae from the knowledge base generated in the previous exploration rounds:

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

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

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

Pack everything into one "theory":

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

Now call the prover:

```
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))  $\forall_{f,a,g,b} (\text{limit}[f, a] \wedge \text{limit}[g, b] \Rightarrow \text{limit}[f + g, a + b]),$

under the assumptions:

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

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

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

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

We assume

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

and show

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

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

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

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

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

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

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

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

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

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

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

We assume

(8)  $\epsilon_0 > 0,$

and show

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

We have to find  $N_2^*$  such that

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

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

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

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

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

We have to find  $\delta_0^*$ ,  $\epsilon_1^*$ , and  $N_2^*$  such that

$$(13) \quad (\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge_n (n \geq N_2^* \Rightarrow |f_0[n] - a_0| < \delta_0^* \wedge |g_0[n] - b_0| < \epsilon_1^*).$$

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

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

which, using (4), is implied by:

$$(\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge_n (n \geq N_2^* \Rightarrow \delta_0^* > 0 \wedge \epsilon_1^* > 0 \wedge n \geq N_0[\delta_0^*] \wedge n \geq N_1[\epsilon_1^*]),$$

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

$$(14) \quad (\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge_n (n \geq N_2^* \Rightarrow \delta_0^* > 0 \wedge \epsilon_1^* > 0 \wedge n \geq \max[N_0[\delta_0^*], N_1[\epsilon_1^*]]).$$

Formula (14) is implied by

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

Partially solving it, formula (15) is implied by

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

Now,

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

can be solved for  $\delta_0^*$  and  $\epsilon_1^*$  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. □

## Problem: Computation of Limits

Find algorithm 'limit' such that

$$\forall_f (\text{is-convergent}[f] \Rightarrow \text{Limit}[f, \text{limit}[f]])$$

This formula is the "specification" of the problem of limit computation.

⏪ ⏩ ⏴ ⏵

16 of 49

## Limit Algorithm: (How Do we Invent this Algorithm?)

We use the collection of knowledge which we proved for the notion Limit and use it as "rewrite rules".

The combination of these rewrite rule is a (first draft) of an algorithm meeting the above specification.

⏪ ⏩ ⏴ ⏵

17 of 49

## Now, Lifting Knowledge to Inferencing

After implementation of the proved knowledge on the **predicate / function** Limit on the meta-level as rules for the **quantifier** *lim*, we can now compute (simplify) for example:

$$\text{Compute}[\lim_{n \rightarrow \infty} [(5 + 1/n) (n / (2n + 5))]]$$

$$\frac{5}{2}$$

$$\text{Compute}[\lim_{a \rightarrow \infty} [(5 + 1/a) (a / (3a + 5))]]$$

$$\frac{5}{3}$$

$$\text{Compute}[\lim_{n \rightarrow \infty} [(1 + 1/n)^n]]$$

$$E$$

(The implementation is, however, not yet proved formally! An important goal for all theory exploration systems designed along these lines!)

What do we have to prove? The application of the *lim* reasoning rules, without knowledge on Limit, have the same effect as using rewriting with the proved knowledge on Limit.

⏪ ⏩ ⏴ ⏵

18 of 49

## The Mathematical Theory Exploration Process

## Automated Mathematical Theory Exploration

## Example: Groebner Bases Algorithm Synthesis

## Conclusion

### Two Ideas for Automating Mathematical Theory Exploration (BB 2002 ff.)

- **Formula Schemes:** Condensed Experience for Inventing Axioms (Definitions), Theorems, Problems, and Methods (Algorithms).
- **Conjectures from Failing Proofs:** Failure has creative power!

### Example: The Monotony Scheme and Limit Theorems

A typical scheme:

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

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

$$\text{Monotony}[\text{Limit}, \oplus, +]$$

i.e.

$$\forall_{a,b,f,g} (\text{Limit}[f, a] \wedge \text{Limit}[g, b]) \Rightarrow \text{Limit}[f \oplus g, a + b]$$

('Monotony' is a "relator" or (the description of) a "category". I cannot discuss the important structuring power of domains, categories, and functors in this talk.)

(I do not discuss the question here how the right substitutions for F and G can be automatically guessed. See, however, section 3 on algorithm synthesis: the creative power of analyzing failing proofs!)



## The Role of Formula Schemes for Invention

By setting up a library of formula schemes, most formulae in the process of exploring a new notion can be generated automatically:

- propositions
- problems
- algorithms.

(Even, many interesting notions can be generated automatically by applying formulae schemes.)

## Conjecture Generation from Failing Proofs

The idea:

- One tries to prove (automatically) a theorem.
- From a failing proof one (automatically) tries to extract an idea for lemmata whose truth may make the proof of the theorem possible.
- One then engages in the (automated) proof of the conjectured lemmata.
- This process may be called recursively ! A "cascade" of conjectures (and, in the positive case, theorems) is automatically generated.

We show the power of this idea in a non-trivial example of automated algorithm invention!

## The Mathematical Theory Exploration Process

### Automated Mathematical Theory Exploration

### Example: Groebner Bases Algorithm Synthesis

### Conclusion

#### What I want to illustrate with this example:

Algorithm synthesis: one of the aspects of mathematical theory exploration.

A combination of

- (automated) theorem **proving**
- (automated) application of formula **schemes** for invention
- (automated) analysis of **failing** proofs

yields a powerful algorithm synthesis method ("lazy thinking method", BB 2002).

Powerful: able to synthesize algorithms for non-trivial problems.

In particular, powerful enough to replace myself.

## "Non-trivial"

Construction of Groebner bases:

- at the time of invention (1965, BB) was conjectured to be algorithmically unsolvable
- dozens of applications in algebraic geometry, invariant theory, optimization, coding theory, cryptography, symbolic summation, geo theorem proving, graph theory, ..., origami proving, sudoku solving, ...
- > 1000 papers, > 10 textbooks, > 3000 citations
- not yet synthesized by other synthesis methods.



25 of 49

## The Algorithm Invention ("Synthesis") Problem

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

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

A general synthesis algorithm cannot exist but ...



26 of 49

## 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.

Our 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.

## Algorithm Synthesis by "Lazy Thinking" (BB 2002)

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

- ♣ We assume we have "complete" **knowledge** on the auxiliary notion appearing in P.
- ♣ Consider known fundamental ideas ("algorithm schemes A") of how to structure algorithms A in terms of subalgorithms B, ...  
Try one scheme A after the other.
- ♣ For the chosen scheme A, try to prove  $\forall_x P[x, A[x]]$ : From the **failing proof construct specifications** for the subalgorithms B, ... occurring in A.

## Automated Invention of Sufficient Specifications for the Subalgorithms

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

Collect temporary assumptions  $T[x_0, \dots, A[\ ], \dots]$   
and temporary goals  $G[x_0, \dots, B[\dots, A[\ ] \dots]]$   
and produce specification

$$\forall_{x, \dots, y, \dots} (T[x, \dots, Y, \dots] \Rightarrow G[y, \dots, m[Y]]).$$

Details: see papers [BB 2003] and example.

## The method works well on simple problems like sorting

See [Buchberger 2003].

For example, using the divide-and-conquer scheme

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

the method finds (in approx. 2 minutes on a laptop) that all subalgorithms  $S, M, L, R$  satisfying the following specifications make  $A$  a correct sorting algorithm:

⏪ ⏩ ⏴ ⏵

30 of 49

$$\forall_x (\text{is-trivial-tuple}[x] \Rightarrow S[x] = x)$$

$$\forall_{y,z} \left( \begin{array}{l} \text{is-sorted}[y] \\ \text{is-sorted}[z] \end{array} \Rightarrow \begin{array}{l} \text{is-sorted}[M[y, z]] \\ M[y, z] \approx (y \times z) \end{array} \right)$$

$$\forall_x (L[x] \times R[x] \approx x)$$

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

Now we can continue, recursively, with synthesizing - or retrieving - algorithms  $S, M, L, R$  satisfying the above specifications.

⏪ ⏩ ⏴ ⏵

31 of 49

## How Far Can We Go With Lazy Thinking

Successful synthesis of Groebner bases algorithm. See:

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.

⏪ ⏩ ⏴ ⏵

32 of 49

## The Problem of Constructing Gröbner Bases

Find algorithm `Gb` such that

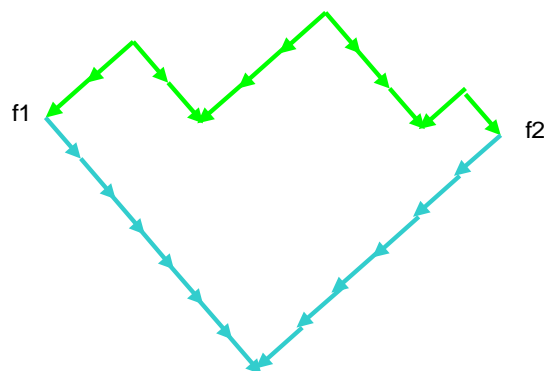
$$\forall \text{is-finite}[F] \left( \begin{array}{l} \text{is-finite}[\text{Gb}[F]] \\ \text{is-Gröbner-basis}[\text{Gb}[F]] \\ \text{ideal}[F] = \text{ideal}[\text{Gb}[F]]. \end{array} \right)$$

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

$\rightarrow_G$  ... a division step.

## Confluence of Division $\rightarrow_G$

$$\text{is-confluent}[\rightarrow] : \Leftrightarrow \forall_{f_1, f_2} (f_1 \leftrightarrow^* f_2 \Rightarrow f_1 \downarrow^* f_2)$$



## The Essential Algorithmic Idea in Groebner Bases Theory (BB 1965)

It suffices to consider the reduction of

```
least-common-multiple[lp[g1], lp[g2]]
```

for all polynomials  $g_1$  and  $g_2$  in the basis  $F$ .

⏪ ⏩

35 of 49

## Hence, the Essential Methodologic Question for the Power of Algorithm Synthesis

Can we *automatically produce the idea* (and can we automatically prove the idea correct) that

```
least-common-multiple[lp[g1], lp[g2]]
```

are the essential objects we have to consider.

So let's start with the synthesis using the "lazy thinking" method.

⏪ ⏩

36 of 49

## We Assume that we Have "Complete" Knowledge on the Auxiliary Concepts Involved

```
h1 →G h2 ⇒ p . h1 →G p . h2
```

etc.

⏪ ⏩

37 of 49

## Use Algorithm Schemes

For example: a scheme for any domain, in which we have a reduction operation

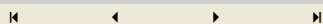
$\text{rd}[f, g]$  (result of "reducing  $f$  by  $g$ ") satisfying  $\text{rd}[f, g] \preceq f$

w.r.t. some Noetherian ordering  $\preceq$ .

$$\forall_{A, \text{lc}, \text{df}} \text{pair-completion}[A, \text{lc}, \text{df}] \Leftrightarrow$$

$$\begin{aligned} & \forall_{\mathbb{F}} A[\mathbb{F}] = A[\mathbb{F}, \text{pairs}[\mathbb{F}]] \\ & \forall_{\mathbb{F}} A[\mathbb{F}, \langle \rangle] = \mathbb{F} \\ & \forall_{\mathbb{F}, g_1, g_2, \bar{p}} A[\mathbb{F}, \langle \langle g_1, g_2 \rangle, \bar{p} \rangle] = \\ & \quad \text{where } [f = \text{lc}[g_1, g_2], \\ & \quad \quad h_1 = \text{trd}[\text{rd}[f, g_1], \mathbb{F}], h_2 = \text{trd}[\text{rd}[f, g_2], \mathbb{F}], \\ & \quad \left. \begin{array}{l} A[\mathbb{F}, \langle \bar{p} \rangle] \\ A[\mathbb{F} - \text{df}[h_1, h_2], \\ \langle \bar{p} \rangle \times \langle \langle \mathbb{F}_k, \text{df}[h_1, h_2] \rangle_{k=1, \dots, |\mathbb{F}|} \rangle] \end{array} \right\} \begin{array}{l} \Leftarrow h_1 = h_2 \\ \Leftarrow \text{otherwise} \end{array} \end{array} \end{aligned}$$

(What would happen, if we started with another algorithm scheme, e.g. divide-and-conquer?)



38 of 49

## 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.)



39 of 49

## Details

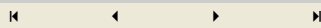
First, it can be proved (independent of what  $\text{lc}$  and  $\text{df}$  are), that if the algorithm terminates, the final result is a finite set (of polynomials)  $G$  that has the property

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

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.



40 of 49

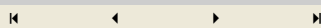
## Using Available Knowledge

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

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

Newman's lemma (1942):

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



41 of 49

## The (Automated) Proof Attempt

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

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

We have to find a polynomial  $g$  such that

$$\begin{cases} f1 \rightarrow_G^* g, \\ f2 \rightarrow_G^* g. \end{cases}$$

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

```

lp[g1] | p,
f1 = rd[p, g1],
lp[g2] | p,
f2 = rd[p, g2].

```

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

$$\bigvee \left\{ \begin{array}{l} h1 = h2 \\ df[h1, h2] \in G, \end{array} \right.$$

where

```

h1 := trd[f1', G], f1' := rd[lc[g1, g2], g1],
h2 := trd[f2', G], f2' := rd[lc[g1, g2], g2].

```

⏪

⏩

⏪

⏩

42 of 49

## 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 that  $lc[g1, g2]$  is reducible w.r.t.  $g1$  and  $g2$ . The other cases are easy.)

Hence, by elementary properties of polynomial reduction,

$$\bigvee_{a,q} ( a \ q \ lc[g1, g2] \rightarrow_{g1} a \ q \ rd[lc[g1, g2], g1] \rightarrow_{G^*} a \ q \ trd[rd[lc[g1, g2], g1], G] = a \ q \ trd[rd[lc[g1, g2], g2], G] \leftarrow_{G^*} a \ q \ rd[lc[g1, g2], g2] \leftarrow_{g2} a \ q \ lc[g1, g2] ).$$

Now we are stuck in the proof.

⏪

⏩

⏪

⏩

43 of 49

## 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

$$\bigvee_{p, g1, g2} \left( \left( \left\{ \begin{array}{l} lp[g1] | p \\ lp[g2] | p \end{array} \right\} \right) \Rightarrow \left( \exists_{a,q} (p = a \ q \ lc[g1, g2]) \right) \right), \quad (lc \text{ requirement})$$

With such an  $lc$ , we then would have

$$p \rightarrow_{g_1} rd[p, g_1] = a \ q \ rd[lc[g_1, g_2], g_1] \rightarrow_G^* a \ q \ trd[rd[lc[g_1, g_2], g_1], G] = \\ a \ q \ trd[rd[lc[g_1, g_2], g_2], G] \leftarrow_G^* a \ q \ rd[lc[g_1, g_2], g_2] = rd[p, g_2] \leftarrow_{g_2} p$$

and, hence,

$$f1 \rightarrow_G^* a \ q \ trd[rd[lc[g_1, g_2], g_1], G],$$

$$f2 \rightarrow_G^* a \ q \ trd[rd[lc[g_1, g_2], g_1], G],$$

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



44 of 49

## Summarize the (Automatically Generated) Specifications of the Subalgorithm $lc$

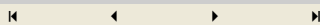
Using the above specification generation rule, we see that we could proceed successfully with the proof if  $lc[g_1, g_2]$  satisfied the following requirement

$$\forall_{p, g_1, g_2} \left( \left( \begin{array}{l} lp[g_1] \mid p \\ lp[g_2] \mid p \end{array} \right) \Rightarrow (lc[g_1, g_2] \mid p) \right),$$

and the requirements:

$$lp[g_1] \mid lc[g_1, g_2], \\ lp[g_2] \mid lc[g_1, g_2].$$

Now this problem can be attacked independently of any Gröbner bases theory, ideal theory etc. In fact, it can be solved by high-school mathematics!



45 of 49

## A Suitable $lc$

$$lc_p[g_1, g_2] = lcm[lp[g_1], lp[g_2]]$$

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!

⏪ ⏩

46 of 49

## Case $h1 \neq h2$

In this case,  $df[h1, h2] \in G$ :

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

$$\forall_{h1, h2} (h1 \downarrow_{\{df[h1, h2]\}} * h2) \quad (\text{df requirement}).$$

⏪ ⏩

47 of 49

## 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 satisfies (df requirement), namely

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

because, in fact,

$$\forall_{f, g} (f \downarrow_{\{f-g\}} * g).$$

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

⏪ ⏩

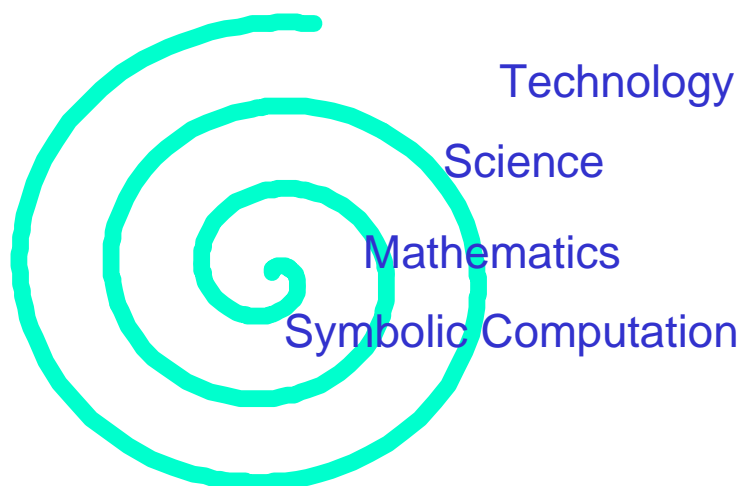
48 of 49

## The Mathematical Theory Exploration Process

## Automated Mathematical Theory Exploration

## Example: Groebner Bases Algorithm Synthesis

## Conclusion



The automation of the mathematical invention and verification process  
is in the center of the  
past, present, and future scientific / technologic spiral.

# 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.