

Algorithm Synthesis in Theorema: Case Study Gröbner Bases

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

Talk at University of Edinburgh, June 23, 2005

The Theorema Project

An "Algorithm" for Algorithm Synthesis

Synthesis of a Gröbner Bases Algorithm

Conclusion

The Theorema Project

An "Algorithm" for Algorithm Synthesis

Synthesis of a Gröbner Bases Algorithm

Conclusion

The Objective of Theorema

A system that aims at supporting the entire "mathematical theory exploration" process:

Starting from some given mathematical concepts and mathematical knowledge on these concepts [within a uniform logical language \(predicate logic\)](#),

- invent [definitions](#) (axioms) of new concepts
- invent and prove / disprove [propositions](#) on these concepts
- invent [problems](#)
- invent and verify [algorithms](#) for problems
- store the definitions, propositions, problems, algorithms in [structured knowledge libraries](#)

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

Emphasis of this Talk

(Partially) automated invention of algorithms from problem specifications.

⏪ ⏩

6 of 48

A Simple Example of the Theorema Proof Style

```
Definition["addition", any[m, n],
  m + 0 = m      " +0:"
  m + n + 1 = (m + n) + 1 " +.:" ]
```

```
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}}];
```

⏪ ⏩

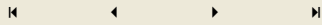
7 of 48

Example: Failing Proof

```
Proposition["commutativity of addition", any[m, n],
  m + n = n + m " + = " ]
```

```
Prove[Proposition["commutativity of addition"],
  using → ⟨Definition["addition"]⟩,
  by → NNEqIndProver,

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



8 of 48

Example: (Automatic) Inventions from Failing Proofs

Theorema tool "Cascade":

```
Prove[Proposition["commutativity of addition"],
  using → Definition["addition"],
  by → Cascade[NNEqIndProver, ConjectureGenerator],
  ProverOptions → {TermOrder → LeftToRight}];
```

This idea (in a slight generalization) is also an essential ingredient for algorithm synthesis in *Theorema*.



9 of 48

Other Provers in Theorema

Predicate logic: natural deduction, S-decomposition

Elementary analysis: PCS (alternating quantifiers)

Set theory

Induction on natural numbers, on tuples

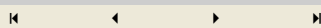
Functors

Equality, sequence variables

Combinatorial identities

Geometry (based on algebraic methods like Gröbner bases)

...



10 of 48

The Theorema Project

An Procedure for Algorithm Synthesis

Synthesis of a Gröbner Bases Algorithm

Conclusion



11 of 48

The Algorithm Invention ("Synthesis") Problem

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

$$\forall_x P[x, A[x]].$$

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 ...



12 of 48

Literature

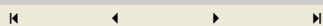
There is a rich literature on algorithm synthesis methods, see survey

[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, of course, lots of similarities with what you are doing (critics, etc.)



13 of 48

The "Lazy Thinking" Method for Algorithm Synthesis (BB 2001): Intuition

"Lazy": Delay invention until necessary.

First idea: Use available "algorithmic ideas":

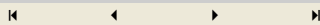
An algorithmic idea: A way how to reduce the solution A of a problem to the solution B, ... of other problems.

The reduction can be expressed by an "algorithm scheme": $A[x] = \dots A[\dots] \dots B[\dots] \dots$

Second idea: Learn from failing proof:

For the chosen scheme: Try to prove $\forall_x P[x, A[x]]$.

From the failing proof read off conditions on B,



14 of 48

Some Comments on the Method

The lazy thinking method is **simple** and **natural** (but powerful).

The lazy thinking method is both an **algorithm methodology** and an approach for **automated algorithm synthesis**.

The lazy thinking method **can be based on any automated reasoning system** that satisfies certain (natural) conditions:

Access to proof objects, in particular to failing proof objects.

Proofs in "natural style".

The "Lazy Thinking" Method for Algorithm Synthesis: Sketch

Given a problem specification P

- consider various "algorithm schemes" for A, e.g.

$$A[\langle \rangle] = \mathbf{c}$$

$$\forall_{\mathbf{x}} A[\langle \mathbf{x} \rangle] = \mathbf{s}[\langle \mathbf{x} \rangle]$$

$$\forall_{\mathbf{x}, \bar{\mathbf{y}}} (A[\langle \mathbf{x}, \mathbf{y}, \bar{\mathbf{z}} \rangle] = \mathbf{i}[\mathbf{x}, \mathbf{y}, A[\langle \mathbf{y}, \bar{\mathbf{z}} \rangle]])$$
- and try to **prove (automatically)** $\forall_{\mathbf{x}} P[\mathbf{x}, A[\mathbf{x}]]$.
- This proof will normally **fail** because nothing is known on the unspecified sub-algorithms c, s, i, ... in the algorithm scheme.
- From the temporary assumptions and goals in the failing proof situation **(automatically) generate such specifications for the unspecified sub-algorithms** c, s, i, ... that would make the proof possible.

Now, apply the method recursively to the auxiliary functions.

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

Problem: Synthesize "sorted" such that

$$\forall_{\mathbf{x}} \text{is-sorted-version}[\mathbf{x}, \text{sorted}[\mathbf{x}]].$$

("Correctness Theorem")

Knowledge on Problem:

$$\forall_{\mathbf{x}, \mathbf{y}} \left(\text{is-sorted-version}[\mathbf{x}, \mathbf{y}] \Leftrightarrow \begin{array}{l} \text{is-sorted}[\mathbf{y}] \\ \text{is-permuted-version}[\mathbf{x}, \mathbf{y}] \end{array} \right)$$

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

$$\forall_{\mathbf{x}} \text{is-sorted}[\langle \mathbf{x} \rangle]$$

$$\forall_{\mathbf{x}, \mathbf{y}, \bar{\mathbf{z}}} \left(\text{is-sorted}[\langle \mathbf{x}, \mathbf{y}, \bar{\mathbf{z}} \rangle] \Leftrightarrow \begin{array}{l} \mathbf{x} \geq \mathbf{y} \\ \text{is-sorted}[\langle \mathbf{y}, \bar{\mathbf{z}} \rangle] \end{array} \right)$$

etc.

An Algorithm Scheme: Divide and Conquer

$$\forall_x \left(\text{sorted}[x] = \begin{cases} \text{special}[x] & \Leftarrow \text{is-trivial-tuple}[x] \\ \text{merge}[\text{sorted}[\text{left-split}[x]], \text{sorted}[\text{right-split}[x]]] & \Leftarrow \text{otherwise} \end{cases} \right)$$

`special`, `merge`, `left-split`, `right-split` are unknowns.

We Now Start Proving the Correctness Theorem and Analyze the Failing Proof: see notebooks with failing proofs.

⏪

⏩

⏪

⏩

17 of 48

Automated Invention of Sufficient Specifications for the Subalgorithms

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

Collect temporary assumptions $T[x_0, \dots, A[\], \dots]$

and temporary goals $G[x_0, \dots, m[A[\]]]$

and produces specification

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

Details: see papers [BB 2003] and example.

⏪

⏩

⏪

⏩

18 of 48

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):

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

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

$$\forall_x (\text{left-split}[x] \times \text{right-split}[x] \approx x)$$

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

⏪ ⏩ ⏴ ⏵

19 of 48

What Do We Have Now?

- **Case A:** We find algorithms **S, M, L, R** in our knowledge base for which the properties specified above for **special, merged, left-split, right-split** 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 **S, M, L, R** in our knowledge base.

In this case, we apply Lazy Thinking again in order to synthesize appropriate **special, merged, left-split, right-split**

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).

⏪ ⏩ ⏴ ⏵

20 of XXX

Example: Synthesis of Insertion-Sort

Synthesize A such that

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

Algorithm Scheme: "simple recursion"

$$\begin{aligned} A[\langle \rangle] &= \mathbf{c} \\ \forall_{\mathbf{x}} A[\langle \mathbf{x} \rangle] &= \mathbf{S}[\langle \mathbf{x} \rangle] \\ \forall_{\mathbf{x}, \bar{\mathbf{y}}} (A[\langle \mathbf{x}, \bar{\mathbf{y}} \rangle] &= \mathbf{i}[\mathbf{x}, A[\langle \bar{\mathbf{y}} \rangle]]) \end{aligned}$$

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

$$\begin{aligned} \mathbf{c} &= \langle \rangle \\ \forall_{\mathbf{x}} (\mathbf{S}[\langle \mathbf{x} \rangle] &= \langle \mathbf{x} \rangle) \\ \forall_{\mathbf{x}, \bar{\mathbf{y}}} \left(\text{is-sorted}[\langle \bar{\mathbf{y}} \rangle] \Rightarrow \right. & \left. \begin{aligned} &\text{is-sorted}[\mathbf{i}[\mathbf{x}, \langle \bar{\mathbf{y}} \rangle]] \\ &\mathbf{i}[\langle \mathbf{x}, \bar{\mathbf{y}} \rangle] \approx (\mathbf{x} \sim \langle \bar{\mathbf{y}} \rangle) \end{aligned} \right) \end{aligned}$$

⏪ ⏩ ⏴ ⏵

21 of 48

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.

⏪ ⏩ ⏴ ⏵

22 of 48

The Problem of Constructing Gröbner Bases is Non-trivial

[Dozens of fundamental problems](#) in algebraic geometry, invariant theory, optimization, graph theory, coding theory, cryptography, statistics, symbolic summation, symbolic solution of differential equations, ... [can be reduced](#) to the construction of Gröbner bases. (Approx. 500 papers on the application of the Gröbner bases method.)

Some of these problems were [open for decades](#).

[Main algorithmic idea](#) of Gröbner bases theory: The "S-polynomials" together with the S-polynomial theorem.

(B.B. An Algorithmical Criterion for the Solvability of Algebraic Systems of Equations. *Aequationes mathematicae* 4/3, 1970.)

Hence, question: Can Lazy Thinking [automatically invent the notion of S-polynomial](#) and automatically deliver the S-polynomial theorem?

⏪ ⏩ ⏴ ⏵

23 of 48

The Gröbner Bases Algorithm in *Mathematica*, Maple, Derive, ...

```
f1 = x y - 2 y z - z - 1;
f2 = y2 - x2 z + x z + 2;
f3 = z2 - y2 x + x + 1;

F = {f1, f2, f3};
```

```
{time, G} = GroebnerBasis[F] // Timing
```

```
{0.01 Second,
{-11 - 23 z - 14 z2 + 177 z3 + 134 z4 + 230 z5 - 383 z6 - 11 z7 - 744 z8 + 5 z9 -
348 z10 + 176 z11 - 64 z12 + 64 z13, 8370016703419 - 97318774930576 y +
207351025151780 z - 722400840178486 z2 + 155529319830493 z3 -
1676870964046611 z4 + 2617008001482037 z5 - 1102382418002270 z6 +
3982342741210737 z7 - 760100969952625 z8 + 2005641117585932 z9 -
1006208474754864 z10 + 397148351653504 z11 - 304335002718272 z12,
-7357801284994 - 12164846866322 x - 24858058005413 z -
54253437844244 z2 - 25982455463515 z3 + 137954693662980 z4 +
82182910700209 z5 + 274904331275643 z6 +
68396806233557 z7 + 121808189089485 z8 - 38565643382556 z9 +
8317571394992 z10 - 19345137946880 z11 - 4858008691392 z12}}
```

⏪ ⏩ ⏴ ⏵

24 of 48

The S-Polynomials

```
S-polynomial[x y - 2 y z - z - 1, y2 - x2 z + x z + 2] =
x z (x y - 2 y z - z - 1) + y (y2 - x2 z + x z + 2)
```

```
x z (x y - 2 y z - z - 1) + y (y2 - x2 z + x z + 2) // Expand
```

```
2 y + y3 - x z + x y z - x z2 - 2 x y z2
```

```

S-polynomial[g1, g2] =
  form the
  least-common-multiple[leading-power-product[g1], leading-power-product[g2]]
  and then ...

```

⏪ ⏩

25 of 48

The Theorema Project

An "Algorithm" for Algorithm Synthesis

Synthesis of a Gröbner Bases Algorithm

Conclusion

⏪ ⏩

26 of 48

The Problem of Constructing Gröbner Bases

Find algorithm `Gb` such that

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

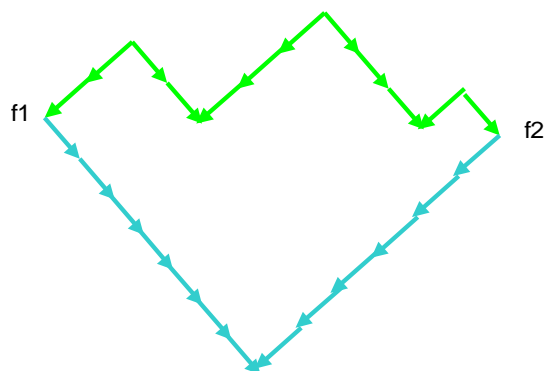
Definitions [BB 1965] :

`is-Gröbner-basis`[G] \Leftrightarrow `is-confluent`[\rightarrow_G].

\rightarrow_G ... a division step.

Confluence of Division \rightarrow_G

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



Knowledge on the Concepts Involved

$h_1 \rightarrow_G h_2 \Rightarrow p \cdot h_1 \rightarrow_G p \cdot h_2$

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⟩ ≃ ⟨⟨Fk, 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 $>$ which interacts with rd in the following natural way:

$$\forall_{f, g} (f > rd[f, g]).$$

30 of 48

The Essential Problem

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

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

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.

31 of 48

Now Start the (Automated) Correctness Proof

With current theorem proving technology, in the *Theorema* system (and other provers), the proof attempt could be done automatically. (Not yet fully implemented.)

« ‹ › »

32 of 48

Details

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

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

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.

« ‹ › »

33 of 48

Using Available Knowledge

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

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

« ‹ › »

34 of 48

The (Automated) Proof Attempt

Let now the power product p and the polynomials f_1, f_2 be arbitrary but fixed and assume

$$\begin{cases} p \rightarrow_G f_1 \\ p \rightarrow_G f_2. \end{cases}$$

We have to find a polynomial g such that

$$\begin{cases} f_1 \rightarrow_{G^*} g, \\ f_2 \rightarrow_{G^*} g. \end{cases}$$

From the assumption we know that there exist polynomials g_1 and g_2 in G such that

$$\begin{cases} lp[g_1] \mid p, \\ f_1 = rd[p, g_1], \\ lp[g_2] \mid p, \\ f_2 = rd[p, g_2]. \end{cases}$$

From the final situation in the algorithm scheme we know that for these g_1 and g_2

$$\bigvee \begin{cases} h_1 = h_2 \\ df[h_1, h_2] \in G, \end{cases}$$

where

$$\begin{cases} h_1 := trd[f_1', G], f_1' := rd[lc[g_1, g_2], g_1], \\ h_2 := trd[f_2', G], f_2' := rd[lc[g_1, g_2], g_2]. \end{cases}$$



Case $h_1=h_2$

$$\begin{aligned} lc[g_1, g_2] \rightarrow_{g_1} rd[lc[g_1, g_2], g_1] \rightarrow_{G^*} trd[rd[lc[g_1, g_2], g_1], G] = \\ trd[rd[lc[g_1, g_2], g_2], G] \leftarrow_{G^*} rd[lc[g_1, g_2], g_2] \leftarrow_{g_2} lc[g_1, g_2]. \end{aligned}$$

(Note that here we used the requirements $rd[lc[g_1, g_2], g_1] < lc[g_1, g_2]$ and $rd[lc[g_1, g_2], g_2] < lc[g_1, g_2]$.)

Hence, by elementary properties of polynomial reduction,

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

Now we are stuck in the proof.



36 of 48

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

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

With such an lc , we then would have

$$p \rightarrow_{g1} rd[p, g1] = a \ q \ rd[lc[g1, g2], g1] \rightarrow_G^* a \ q \ trd[rd[lc[g1, g2], g1], G] = a \ q \ trd[rd[lc[g1, g2], g2], G] \leftarrow_G^* a \ q \ rd[lc[g1, g2], g2] = rd[p, g2] \leftarrow_{g2} p$$

and, hence,

$$f1 \rightarrow_G^* a \ q \ trd[rd[lc[g1, g2], g1], G],$$

$$f2 \rightarrow_G^* a \ q \ trd[rd[lc[g1, g2], g1], G],$$

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



37 of 48

Summarize the (Automatically Generated) Specifications of the Subalgorithm lc

(lc requirement), which also could be written in the form:

$$\forall_{p,g1,g2} \left(\left(\left\{ \begin{array}{l} lp[g1] \mid p \\ lp[g2] \mid p \end{array} \right\} \right) \Rightarrow (lc[g1, g2] \mid p) \right),$$

and the requirements:

```
rd[lc[g1, g2], g1] < lc[g1, g2],
rd[lc[g1, g2], g2] < lc[g1, g2],
```

which, in the case of the domain of polynomials, are equivalent to

```
lp[g1] | lc[g1, g2],
lp[g2] | lc[g1, g2].
```

⏪ ⏩

38 of 48

A Suitable lc

```
lcp[g1, g2] = lcm[lp[g1], 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!

⏪ ⏩

39 of 48

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

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

⏪ ⏩

40 of 48

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,

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

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



41 of 48

The Theorema Project

An "Algorithm" for Algorithm Synthesis

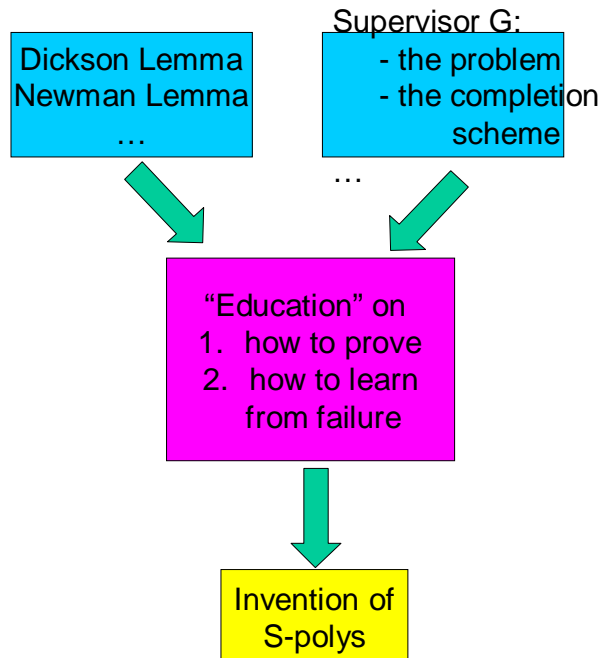
Synthesis of a Gröbner Bases Algorithm

Conclusion



42 of 48

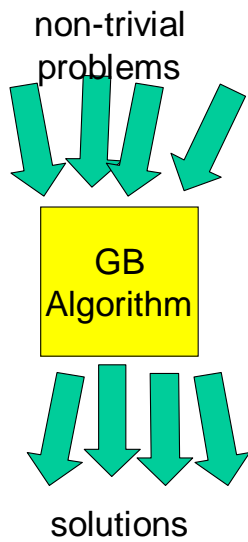
A way of looking at it ("what would have happened if ..."):



Pairs idea?

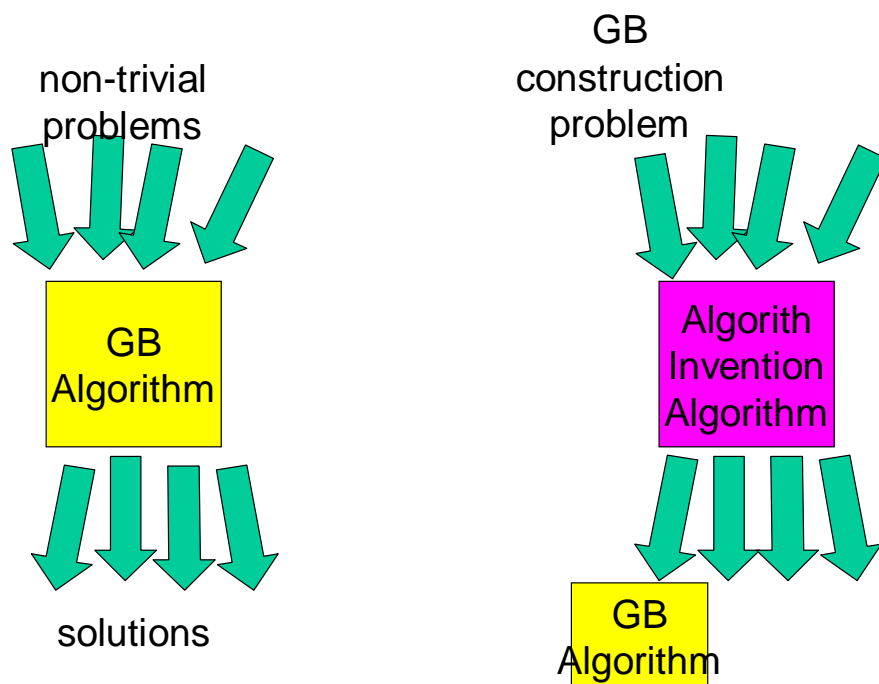
$$\begin{aligned}
 & A[F, \langle \langle g1, g2 \rangle, \bar{p} \rangle] = \\
 & \text{where } [f = \text{lc}[g1, g2], h1 = \text{trd}[\text{rd}[f, g1], F], h2 = \text{trd}[\text{rd}[f, g2], F], \\
 & \left\{ \begin{array}{ll} A[F, \langle \bar{p} \rangle] & \Leftarrow h1 = h2 \\ A[F - \text{df}[h1, h2], \langle \bar{p} \rangle \times \langle \langle F_k, \text{df}[h1, h2] \rangle_{k=1, \dots, |F|}] & \Leftarrow \text{otherwise} \end{array} \right.]
 \end{aligned}$$

Bad and Good for Me



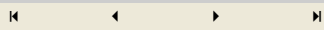
Good for me !

Bad and Good for Me



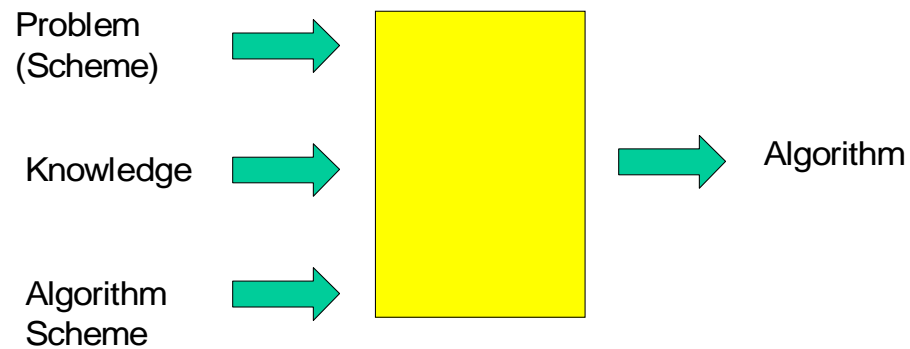
Good for me !

Good and bad for me !



45 of 48

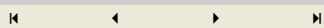
Research Topics



- **Libraries** of algorithm **schemes**.

More generally, libraries of formulae schemes for definitions, propositions, problems, and algorithms.

- **Case studies** of problem (schemes), knowledge, algorithm schemes and how they produce algorithms.
- How well are **current reasoning systems suited** for supporting this approach to algorithm synthesis?
- Improved **algorithms for generating problem specifications** from failing proofs.



46 of 48

Special Semester on Gröbner Bases at RICAM and RISC (Feb - July 2006)



Softwarepark Hagenberg



48 of 48

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 2004]

B. Buchberger.

Towards the Automated Synthesis of a Gröbner Bases Algorithm.

RACSAM (Review of the Royal Spanish Academy of Science), Vol. 98/1, to appear, 10 pages.