# From Gröbner Bases to Automated Theorem Proving and Back

Bruno Buchberger

Lectures at School on Gröbner Bases and Applications
IASBS (Institute for Advanced Studies in Basic Sciences) , Zanjan, Iran,
July 9-22, 2005

**Copyright Bruno Buchberger 2005.**

Copying and storing is allowed under the following conditions:

- The file including the copyright note is kept unchanged

- You send an e-mail to bruno.buchberger@jku.at

- If you use material from this file,please,cite this lecture appropriately.

## An Application of Gröbner Bases: Algorithmic Theorem Proving in Geometry

## Systematic (Algorithmic) Invention of Algorithms

## Systematic Invention of the Gröbner Basis Algorithm

## An Application of Gröbner Bases: Algorithmic Theorem Proving in Geometry

## Systematic (Algorithmic) Invention of Algorithms

## Systematic Invention of the Gröbner Basis Algorithm



## Reduction of the Geo Proving Problem to Gröbner bases Computation

Geo Theorem $\longrightarrow$ ( by coordinatization )

$\forall_{x,y,\ldots}$ ( poly1(x,y,...)=0 $\wedge$ ... $\Rightarrow$ poly(x,y,...)=0 ) $\longrightarrow$

$\neg \exists_{x,y,\ldots}$ ( poly1(x,y,...)=0 $\wedge$ ... $\wedge$ poly(x,y,...)$\neq$0 ) $\longrightarrow$

$\neg \exists_{x,y,\ldots,a}$ ( poly1(x,y,...)=0 $\wedge$ ... $\wedge$ a . poly(x,y,...) - 1 = 0 )

The latter question can be decided by the Gröbner basis method!

The method is implemented, by J. Robu, in the Theorema System:

J. Robu. Algebraic Methods for Automated Theorem Proving in Geometry. PhD. Thesis, RISC, Johannes Kepler University, Linz, Austria, 2002.

B. B., C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, W. Windsteiger. The Theorema Project: A Progress Report. In: 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), M. Kerber and M. Kohlhase (eds.), A.K. Peters, Natick, Massachusetts, ISBN 1-56881-145-4, pp. 98-113.

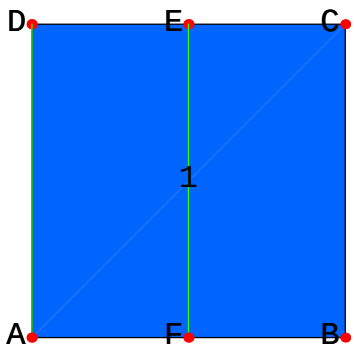


## Very Easy Example: Proving a Property of an Origami Constructions

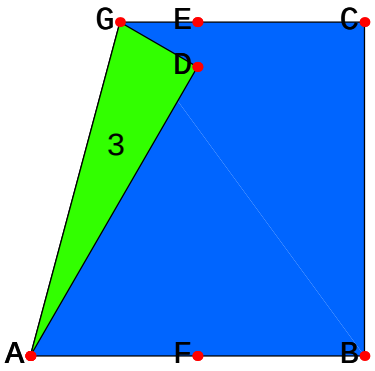
(This is joint work with Tetsuo Ida, Tsukuba University)

**Example of an Origami Construction Problem:** Starting from a square A, B, C, D, find a sequence of origami steps such that, finally, we arrive at an equilateral triangle.

A first solution (does not yield maximum edge length; other solution and more complicated examples see **T. Ida, BB, J. Robu et al. 2003 and 2004**):




Then we fix the point A and fold so that point D will lie on line EF. (This is a legal origami operation.)




Now we can do the analogous step with corner C, fixing B and bringing C onto the current position of D. Thus, we obtain an equilateral triangle.

An Example of a Proof Problem: Prove that, for all squares ABCD, $\overline{GD} = 2\,\overline{ED}$.

6 of 55

## The Translation into a Prove Problem on Equalities:

First, note that $\overline{AB} = \overline{BC} = \overline{CD} = \overline{DA}$, since we start from a square. Hence, whenever the length of one of these four edges occurs, we replace it by $\overline{AB}$.

Now observe that

$$\overline{DF}^2 = \overline{AD}^2 - \overline{AF}^2 = \overline{AB}^2 - \left(\overline{AB}/2\right)^2 = 3/4\,\overline{AB}^2.$$

and

$$\overline{GD}^2 = \overline{GE}^2 + \overline{ED}^2 = \left(\overline{DC}/2 - \overline{GD}\right)^2 + \left(\overline{EF} - \overline{DF}\right)^2 = \left(\overline{AB}/2 - \overline{GD}\right)^2 + \left(\overline{AB} - \overline{DF}\right)^2.$$

We want to decide whether, under these assumptions,

$$\overline{GD} = 2\,\overline{ED} = 2\left(\overline{EF} - \overline{DF}\right) = 2\left(\overline{AB} - \overline{DF}\right).$$

For abbreviation, let's write

$$a = \overline{AB},\ b = \overline{GD},\ f = \overline{DF}.$$

Then, what we want to prove is that

$$\forall_{a\neq 0,f,b} \left( \begin{cases} f^2 = 3/4\,a^2 \\ b^2 = (a/2 - b)^2 + (a - f)^2 \end{cases} \Rightarrow (b = 2\,(a - f)) \right)$$

7 of 55

## The Transformation into a Gröbner Bases Construction Problem

The last formula is equivalent to

$$\neg \exists_{a,f,b} \left( \begin{cases} a \neq 0 \\ f^2 = 3/4\,a^2 \\ b^2 = (a/2 - b)^2 + (a - f)^2 \\ b \neq 2\,(a - f) \end{cases} \right),$$

which is equivalent to

$$\underset{a,f,b,\xi,\eta}{\neg\exists}\left(\begin{cases} a\,\eta = 1 \\ f^2 = 3/4\,a^2 \\ b^2 = (a/2-b)^2 + (a-f)^2 \\ (b-2\,(a-f))\,\xi = 1 \end{cases}\right).$$

This question can be decided by computing the (reduced) Gröbner basis

```
GroebnerBasis[
 {a η - 1, f^2 - 3 / 4 a^2, b^2 - (a / 2 - b)^2 - (a - f)^2, (b - 2 (a - f)) ξ - 1}, {ξ, η, b, f, a}]
```

```
{1}
```

and to check whether or not this Gröbner basis is equal to {1}. Since this is the case, we know that the restricted version of the theorem is true.



## The Automated Generation of the "Degeneration Conditions"

Observe what happens if we leave the condition a≠0 out:

$$\underset{a,f,b,\xi}{\neg\exists}\left(\begin{cases} a\,\eta = 1 \\ f^2 = 3/4\,a^2 \\ b^2 = (a/2-b)^2 + (a-f)^2 \\ (b-2\,(a-f))\,\xi = 1 \end{cases}\right).$$

```
GroebnerBasis[{f^2 - 3 / 4 a^2, b^2 - (a / 2 - b)^2 - (a - f)^2, (b - 2 (a - f)) ξ - 1}, {ξ, b, f, a}]
```

```
{a, f^2, -1 + b ξ + 2 f ξ}
```

The original theorem is not true iff this system has a solution, i.e. iff

a = 0, f = 0, b ≠ 0, $\xi \neq 0$, b = 1/ $\xi$.

Thus, it is true if a ≠ 0.

There is quite some literature on the automated generation of the non-degeneration conditions.

## Example: Trisection an Angle by Origami (jointly with T. Ida)

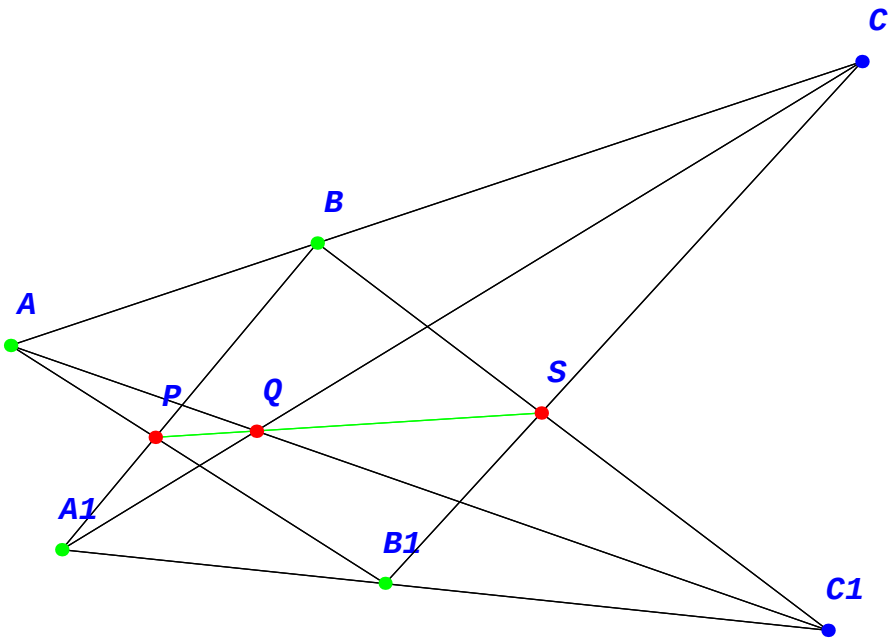
10 of 55

## Automated Proof of Origami Trisection by the Gröbner Bases Method

11 of 55

## Example: Pappus Theorem

- What does the theorem say geometrically?




- Textbook formulation:

  Let A,B, C and A1,B1, C1 be on two lines and P = AB1 ⋂ A1B, Q = AC1 ⋂ A1C, S = BC1 ⋂ B1C. Then P, Q, and S are collinear.

- Input to the system:

```
Proposition["Pappus", any[A, B, A1, B1, C, C1, P, Q, S],
 point[A, B, A1, B1] ∧ pon[C, line[A, B]] ∧ pon[C1, line[A1, B1]] ∧
   inter[P, line[A, B1], line[A1, B]] ∧ inter[Q, line[A, C1], line[A1, C]] ∧
   inter[S, line[B, C1], line[B1, C]] ⇒ collinear[P, Q, S]]
```

- Input to the system:

```
Prove[Proposition["Pappus"], by → GeometryProver,
 ProverOptions → {Method -> "GroebnerProver", Refutation → True}]
```

- Notebook generated automatically by the proving algorithm based on Groebner basis algorithm:

Prove:

(Proposition (Pappus))

$$\forall_{A,B,A1,B1,C,C1,P,Q,S} \left(\text{point}[A, B, A1, B1] \wedge \text{pon}[C, \text{line}[A, B]] \wedge \text{pon}[C1, \text{line}[A1, B1]] \wedge \text{inter}[P, \text{line}[A, B1], \text{line}[A1, B]] \wedge \text{inter}[Q, \text{line}[A, C1], \text{line}[A1, C]] \wedge \text{inter}[S, \text{line}[B, C1], \text{line}[B1, C]] \Rightarrow \text{collinear}[P, Q, S]\right)$$

with no assumptions.

To prove the above statement we shall use the Gröbner basis method. First we have to transform the problem into algebraic form.

Algebraic Form:

To transform the geometric problem into algebraic form we have to chose first an orthogonal coordinate system.

Let's have the origin in point $A$, and points $\{B, C\}$ on the two axes.

Using this coordinate system we have the following points:

$$\{\{A, 0, 0\}, \{B, 0, u_1\}, \{A1, u_2, u_3\}, \{B1, u_4, u_5\}, \{C, 0, u_6\}, \{C1, u_7, x_1\}, \{P, x_2, x_3\}, \{Q, x_4, x_5\}, \{S, x_6, x_7\}\}$$

The algebraic form of the assertion is:

(1)

$$\forall_{x_1,x_2,x_3,x_4,x_5,x_6,x_7} \left(u_3 u_4 + -u_2 u_5 + -u_3 u_7 + u_5 u_7 + u_2 x_1 + -u_4 x_1 == 0 \wedge u_5 x_2 + -u_4 x_3 == 0 \wedge -u_1 u_2 + u_1 x_2 + -u_3 x_2 + u_2 x_3 == 0 \wedge x_1 x_4 + -u_7 x_5 == 0 \wedge -u_2 u_6 + -u_3 x_4 + u_6 x_4 + u_2 x_5 == 0 \wedge u_1 u_7 + -u_1 x_6 + x_1 x_6 + -u_7 x_7 == 0 \wedge -u_4 u_6 + -u_5 x_6 + u_6 x_6 + u_4 x_7 == 0 \Rightarrow x_3 x_4 + -x_2 x_5 + -x_3 x_6 + x_5 x_6 + x_2 x_7 + -x_4 x_7 == 0\right)$$

This problem is equivalent to:

(2)

$$\neg \left(\exists_{x_1,x_2,x_3,x_4,x_5,x_6,x_7} \left(u_3 u_4 + -u_2 u_5 + -u_3 u_7 + u_5 u_7 + u_2 x_1 + -u_4 x_1 == 0 \wedge u_5 x_2 + -u_4 x_3 == 0 \wedge -u_1 u_2 + u_1 x_2 + -u_3 x_2 + u_2 x_3 == 0 \wedge x_1 x_4 + -u_7 x_5 == 0 \wedge -u_2 u_6 + -u_3 x_4 + u_6 x_4 + u_2 x_5 == 0 \wedge u_1 u_7 + -u_1 x_6 + x_1 x_6 + -u_7 x_7 == 0 \wedge -u_4 u_6 + -u_5 x_6 + u_6 x_6 + u_4 x_7 == 0 \wedge x_3 x_4 + -x_2 x_5 + -x_3 x_6 + x_5 x_6 + x_2 x_7 + -x_4 x_7 \neq 0\right)\right)$$

To remove the last inequality, we use the Rabinowitsch trick: Let $v_0$ be a new variable. Then the problem becomes:

(3)

$$\neg \left( \exists_{x_1,x_2,x_3,x_4,x_5,x_6,x_7,v_0} \; (u_3 u_4 + -u_2 u_5 + -u_3 u_7 + u_5 u_7 + u_2 x_1 + -u_4 x_1 == 0 \wedge u_5 x_2 + -u_4 x_3 == 0 \wedge -u_1 u_2 + u_1 x_2 + -u_3 x_2 + u_2 x_3 == 0 \wedge x_1 x_4 + -u_7 x_5 == 0 \wedge -u_2 u_6 + -u_3 x_4 + u_6 x_4 + u_2 x_5 == 0 \wedge u_1 u_7 + -u_1 x_6 + x_1 x_6 + -u_7 x_7 == 0 \wedge -u_4 u_6 + -u_5 x_6 + u_6 x_6 + u_4 x_7 == 0 \wedge 1 + -v_0 \; (x_3 x_4 + -x_2 x_5 + -x_3 x_6 + x_5 x_6 + x_2 x_7 + -x_4 x_7) == 0) \right)$$

This statement is true iff the corresponding Gröbner basis is {1}.

The Gröbner bases is {1}.

Hence, the statement and the original assertion is true.

Statistics:

Time needed to compute the Gröbner bases: 0.42 Seconds.



## Example: Butterfly Theorem

- Textbook formulation:

*A*, *B*, *C* and *D* are four points on circle (*O*). *E* is the intersection of *AC* and *BD*. Through *E* draw a line perpendicular to *OE*, meeting *AD* at *F* and *BC* at *G*. Show that $\overline{FE} = \overline{GE}$.

- Remark:

Using the default coordinate system the algebraic methods are very slow.

- Input to the system:

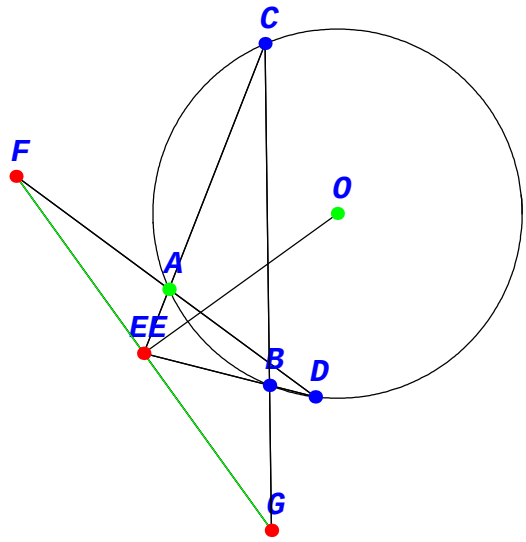
```
Proposition["Butterfly", any[O, A, B, C, D, EE, F, G], point[O, A] ∧
   pon[C, circle[O, A]] ∧
   pon[B, circle[O, A]] ∧
   pon[D, circle[O, A]] ∧
   inter[EE, line[A, C], line[B, D]] ∧
   inter[F, line[D, A], tline[EE, EE, O]] ∧ inter[G, line[B, C], line[EE, F]]
  ⇒ distequal[EE, F, G, EE]]
```

- Input to the system:

```
Simplify[Proposition["Butterfly"], by -> GraphicSimplifier]
```

*A*, *B*, *C* and *D* are four points on circle (*O*). *E* is the intersection of *AC* and *BD*. Through *E* draw a line perpendicular to *OE*, meeting *AD* at *F* and *BC* at *G*. Show that $\overline{FE} = \overline{GE}$.

- Output generated automatically by the system




|*EEF*| ≅ |*GEE*| for this configuration of the points

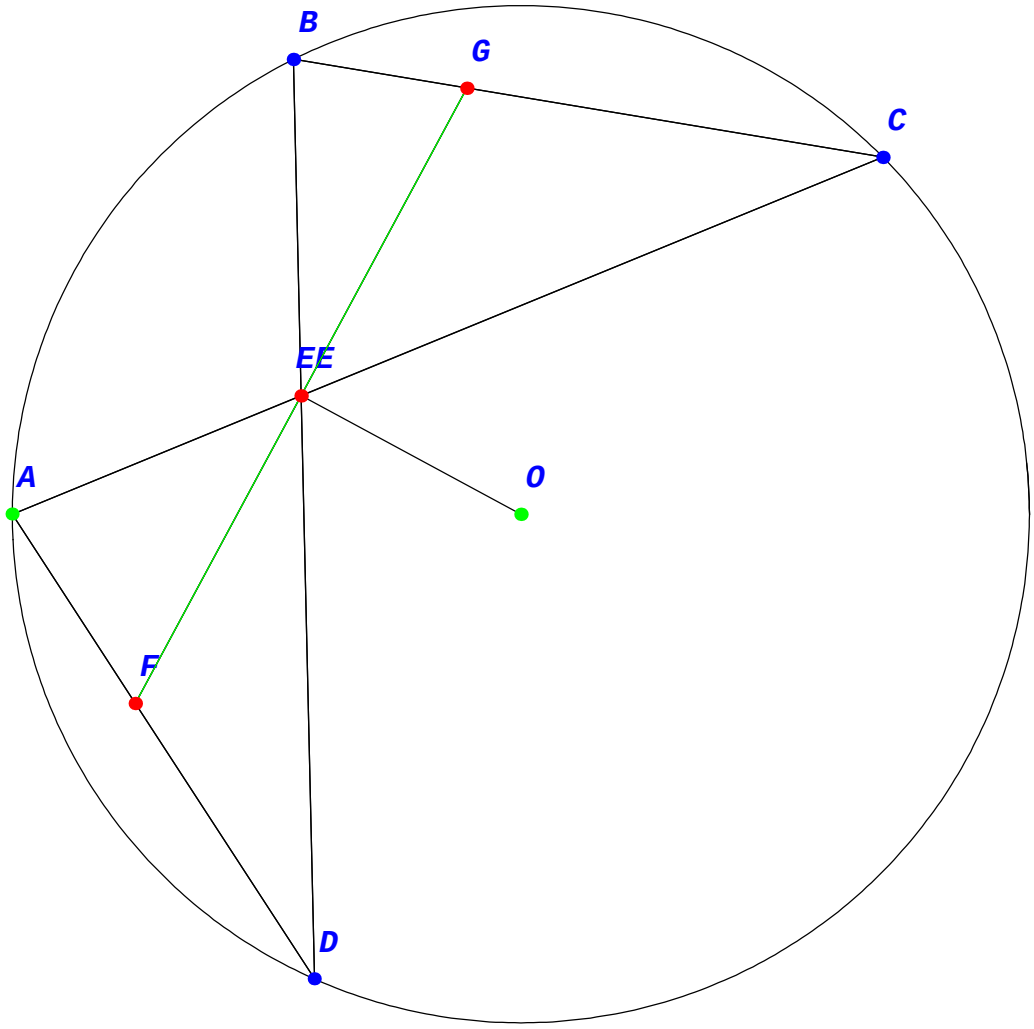
- End of output
- Input to the system:

```
KnowledgeBase["Butterfly_g", any[O, A, B, C, D],
 {{O, {100, 100}}, {A, {0, 100}}, {B, {55, 190}}, {C, {171, 170}}, {D, {60, 10}}}]
```

- Input to the system:

```
Simplify[Proposition["Butterfly"],
 by -> GraphicSimplifier, using → KnowledgeBase["Butterfly_g"]]
```

- Output generated automatically by the system




|EEF| ≅ |GEE| for this configuration of the points

- End of output
- Input to the system:

```
Prove[Proposition["Butterfly"], by → GeometryProver,
 ProverOptions → {Method -> "GroebnerProver", Refutation → False}]
```

- Inserted notebook generated automatically by the system

Prove:

(Proposition (Butterfly))

$$\forall_{O,A,B,C,D,EE,F,G} \left(\text{point}[O, A] \wedge \text{pon}[C, \text{circle}[O, A]] \wedge \text{pon}[B, \text{circle}[O, A]] \wedge \text{pon}[D, \text{circle}[O, A]] \wedge \text{inter}[EE, \text{line}[A, C], \text{line}[B, D]] \wedge \text{inter}[F, \text{line}[D, A], \text{tline}[EE, EE, O]] \wedge \text{inter}[G, \text{line}[B, C], \text{line}[EE, F]] \Rightarrow \text{distequal}[EE, F, G, EE]\right)$$

with no assumptions.

To prove the above statement we shall use the Gröbner bases method. First we have to transform the problem into algebraic form.

Algebraic Form:

To transform the geometric problem into algebraic form we have to chose first an orthogonal coordinate system.

Let's have the origin in point $EE$, and points $\{O\}$ and $\{F, G\}$ on one axix.

Using this coordinate system we have the following points:

$$\{\{EE, 0, 0\}, \{O, u_1, 0\}, \{A, u_2, u_3\}, \{B, u_4, x_1\},\ \{F, 0, x_2\}, \{G, 0, x_3\}, \{C, x_4, x_5\}, \{D, x_6, x_7\}\}$$

The algebraic form of the given construction is:

(1)

$$\forall_{v_1,v_2,x_1,x_2,x_3,x_4,x_5,x_6,x_7} ((-1) + u_4 v_2 + -v_2 x_6 == 0 \bigwedge (-1) + u_2 v_1 + -v_1 x_4 == 0 \bigwedge 2 u_1 u_2 + -u_2^2 + -u_3^2 + -2 u_1 x_4 + x_4^2 + x_5^2 == 0 \bigwedge 2 u_1 u_2 + -u_2^2 + -u_3^2 + -2 u_1 u_4 + u_4^2 + x_1^2 == 0 \bigwedge 2 u_1 u_2 + -u_2^2 + -u_3^2 + -2 u_1 x_6 + x_6^2 + x_7^2 == 0 \bigwedge u_3 x_4 + -u_2 x_5 == 0 \bigwedge x_1 x_6 + -u_4 x_7 == 0 \bigwedge -u_2 x_2 + -u_3 x_6 + x_2 x_6 + u_2 x_7 == 0 \bigwedge u_4 x_3 + x_1 x_4 + -x_3 x_4 + -u_4 x_5 == 0 \Rightarrow x_2^2 + -x_3^2 == 0)$$

We have to compute the Gröbner bases of the hypothesis polynomials (GB).

The polynomials of the Gröbner bases are:

$$\{-u_2^2 + -u_3^2 + (-2 u_1 u_2^2 + 2 u_2^3 + 2 u_2 u_3^2) v_1,\ -2 u_1 u_2 + u_2^2 + u_3^2 + 2 u_1 u_4 + (4 u_1 u_2 u_4 + -2 u_2^2 u_4 + -2 u_3^2 u_4 + -2 u_1 u_4^2) v_2,\ -2 u_1 u_2 u_3 u_4 + u_2^2 u_3 u_4 + u_3^3 u_4 + (2 u_1 u_2^2 + -u_2^3 + -u_2 u_3^2) x_1 + (2 u_1 u_2^2 + -u_2^3 + -u_2 u_3^2 + -u_2^2 u_4 + -u_3^2 u_4) x_2,\ 2 u_1 u_2 u_3 u_4 + -u_2^2 u_3 u_4 + -u_3^3 u_4 + (-2 u_1 u_2^2 + u_2^3 + u_2 u_3^2) x_1 + (2 u_1 u_2^2 + -u_2^3 + -u_2 u_3^2 + -u_2^2 u_4 + -u_3^2 u_4) x_3,\ -2 u_1 u_2^2 + u_2^3 + u_2 u_3^2 + (u_2^2 + u_3^2) x_4,\ 2 u_1 u_2 u_3 + -u_2^2 u_3 + -u_3^3 + (-u_2^2 + -u_3^2) x_5,\ 2 u_1 u_2 u_4 + -u_2^2 u_4 + -u_3^2 u_4 + (2 u_1 u_2 + -u_2^2 + -u_3^2 + -2 u_1 u_4) x_6,\ (-2 u_1 u_2 + u_2^2 + u_3^2) x_1 + (-2 u_1 u_2 + u_2^2 + u_3^2 + 2 u_1 u_4) x_7,\ 2 u_1 u_2 + -u_2^2 + -u_3^2 + -2 u_1 u_4 + u_4^2 + x_1^2\}$$

We compute the normal form of the conclusion polynomial modulo GB.

The normal form is: 0

As the obtained normal form equals to 0 the statement is generically true.

Statistics:

The length of the Gröbner bases is 9

Time needed to compute the Gröbner bases: 0.27 Seconds.

Time needed to compute the normal form of the conclusion polynomial modulo GB: 0.32 Seconds.

□

- End of inserted notebook



An Application of Gröbner Bases: Algorithmic Theorem Proving in Geometry

## Systematic (Algorithmic) Invention of Algorithms

Systematic Invention of the Gröbner Basis Algorithm



## The Theorema System

Theorema 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 system is programmed in *Mathematica*.

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

## Emphasis in these Lectures

(Partially) automated invention of algorithms from problem specifications.



## "Lazy Thinking" Method for Algorithm Synthesis = My Advice to "Students" (= "Computers") How to Invent Algorithms

Given: A problem P.

Find: An algorithm A for P.

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

  Try one scheme A after the other.
- ♣ Try to prove that A solves P: From the failing proof construct specifications of the subalgorithms.



## A Simple Example of an Automatically Generated Proof in Theorema

```
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}}];
```



## A More Complicated Example of a Theorema Proof

```
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 Creative Power of Failing Proofs

Before



## 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}}];
```



## 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*.

## 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)

...



## The Algorithm Invention ("Synthesis") Problem

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

```
∀ P[x, A[x]].
 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 ...

## 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.



## 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] = .... A[ ...] ... B[ ...] ...

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, ....

Q property for B



## Some Comments on the Method

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

The lazy thinking method is both

an algorithm invention methodology or a didactic principle for humans

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".

We implemented the method in Theorema.

## The "Lazy Thinking" Method for Algorithm Synthesis: Sketch

Given a problem specification P

- consider various "algorithm schemes" for A, e.g.
  $\mathbf{A}[\langle\rangle] = \mathbf{c}$
  $\forall_{x} \mathbf{A}[\langle x\rangle] = \mathbf{s}[\langle x\rangle]$
  $\forall_{x,y} (\mathbf{A}[\langle x, y, z\rangle] = \mathbf{i}[x, y, \mathbf{A}[\langle y, z\rangle]])$
- and try to prove (automatically) $\forall_{x} \mathbf{P}[x, \mathbf{A}[x]]$.
- 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

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

("Correctness Theorem")

Knowledge on Problem:

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

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

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

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

etc.



## An Algorithm Scheme: Divide and Conquer

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

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.



## 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

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

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):

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

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

$$\forall_{x} \left(\text{left–split}[x] \asymp \text{right–split}[x] \approx x\right)$$

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



## 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).

## Example: Synthesis of Insertion-Sort

Synthesize A such that

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

Algorithm Scheme: "simple recursion"

$$A[\langle\rangle] = c$$
$$\forall_{x} \; A[\langle x \rangle] = s[\langle x \rangle]$$
$$\forall_{x,y} \; (A[\langle x, y \rangle] = i[x, A[\langle y \rangle]])$$

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

$$c = \langle\rangle$$
$$\forall_{x} \; (s[\langle x \rangle] = \langle x \rangle)$$
$$\forall_{x,y} \left( \text{is-sorted}[\langle y \rangle] \Rightarrow \begin{array}{l} \text{is-sorted}[i[x, \langle y \rangle]] \\ i[\langle x, y \rangle] \approx (x \frown \langle y \rangle) \end{array} \right)$$



## 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.



## 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?



## An Application of Gröbner Bases: Algorithmic Theorem Proving in Geometry

## Systematic (Algorithmic) Invention of Algorithms

## Systematic Invention of the Gröbner Basis Algorithm



# The Problem of Constructing Gröbner Bases

Find algorithm Gb such that

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

Definitions [BB 1965] :

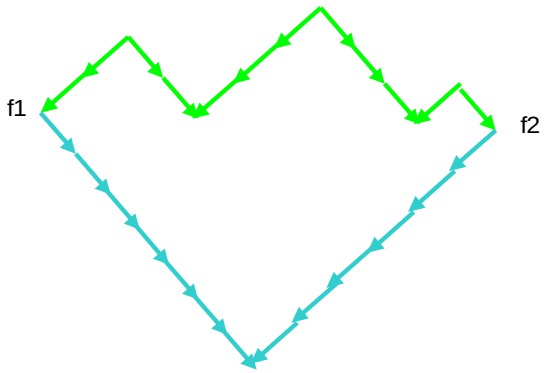
```
is-Gröbner-basis[G] ⇔ is-confluent[ →G ].
```

$\rightarrow_G$ ... a division step.



## Confluence of Division $\rightarrow_G$

```
is-confluent[ → ] : ⇔ ∀f1,f2 (f1 ↔* f2 ⇒ f1↓*f2)
```






## Knowledge on the Concepts Involved

```
h1 →G h2 ⇒ p . h1 →G p . h2
```

etc.

## Algorithm Scheme "Critical Pair / Completion"

$$A[F] = A[F, pairs[F]]$$
$$A[F, \langle\rangle] = F$$

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

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:

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



## The Essential Problem

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

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

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.

## 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.)



## Details

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

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

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

$$\begin{aligned} &\text{is-finite}[G], \\ &\text{ideal}[F] = \text{ideal}[G], \\ &\text{is-Gröbner-basis}[G], \\ &\quad \text{i.e. is-Church-Rosser}[\ \to_G\ ]. \end{aligned}$$

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

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

Newman's lemma (1942):

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

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

$$f1 \downarrow^* f2 \Leftrightarrow \exists_{g} \left( \left\{ \begin{matrix} f1 \to^* g \\ f2 \to^* g \end{matrix} \right. \right)$$



## The (Automated) Proof Attempt

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

$$\left\{ \begin{matrix} p \to_G f1 \\ p \to_G f2. \end{matrix} \right.$$

We have to find a polyonomial g such that

$$\begin{matrix} f1 \to_G^* g, \\ f2 \to_G^* g. \end{matrix}$$

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

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

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

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

where

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

## Case h1=h2

$$\text{lc}[g1, g2] \to_{g1} \text{rd}[\text{lc}[g1, g2], g1] \to_{G}^{*} \text{trd}[\text{rd}[\text{lc}[g1, g2], g1], G] = \text{trd}[\text{rd}[\text{lc}[g1, g2], g2], G] \leftarrow_{G}^{*} \text{rd}[\text{lc}[g1, g2], g2] \leftarrow_{g2} \text{lc}[g1, g2].$$

(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,

$$\forall_{a,q} \left(a\, q\, \text{lc}[g1, g2] \to_{g1} a\, q\, \text{rd}[\text{lc}[g1, g2], g1] \to_{G}^{*} a\, q\, \text{trd}[\text{rd}[\text{lc}[g1, g2], g1], G] = a\, q\, \text{trd}[\text{rd}[\text{lc}[g1, g2], g2], G] \leftarrow_{G}^{*} a\, q\, \text{rd}[\text{lc}[g1, g2], g2] \leftarrow_{g2} a\, q\, \text{lc}[g1, g2]\right).$$

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

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

With such an lc, we then would have

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

and, hence,

$$f1 \to_{G}^{*} a\, q\, \text{trd}[\text{rd}[\text{lc}[g1, g2], g1], G],$$

$$f2 \to_{G}^{*} a\, q\, \text{trd}[\text{rd}[\text{lc}[g1, g2], g1], G],$$

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

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

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

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

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].
```



## 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!



## 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

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

50 of 55

## 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,
```

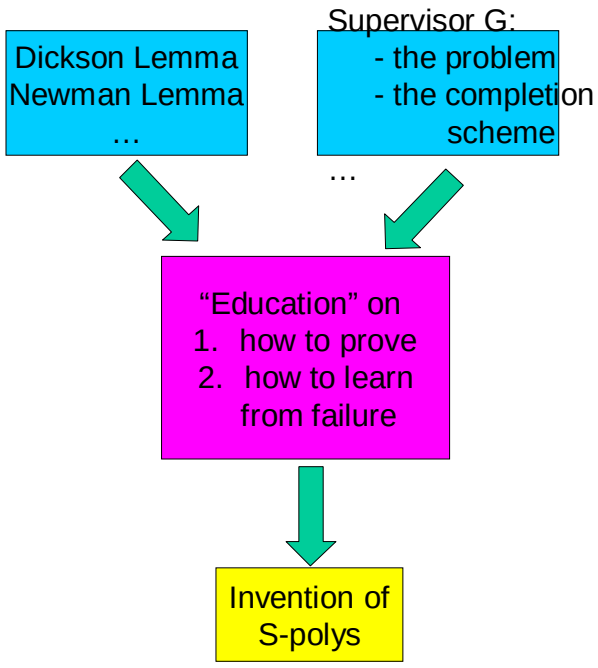
because, in fact,

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

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

51 of 55

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

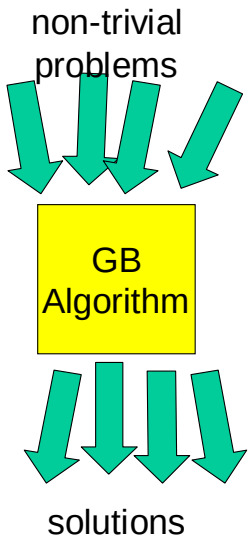


Pairs idea?

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

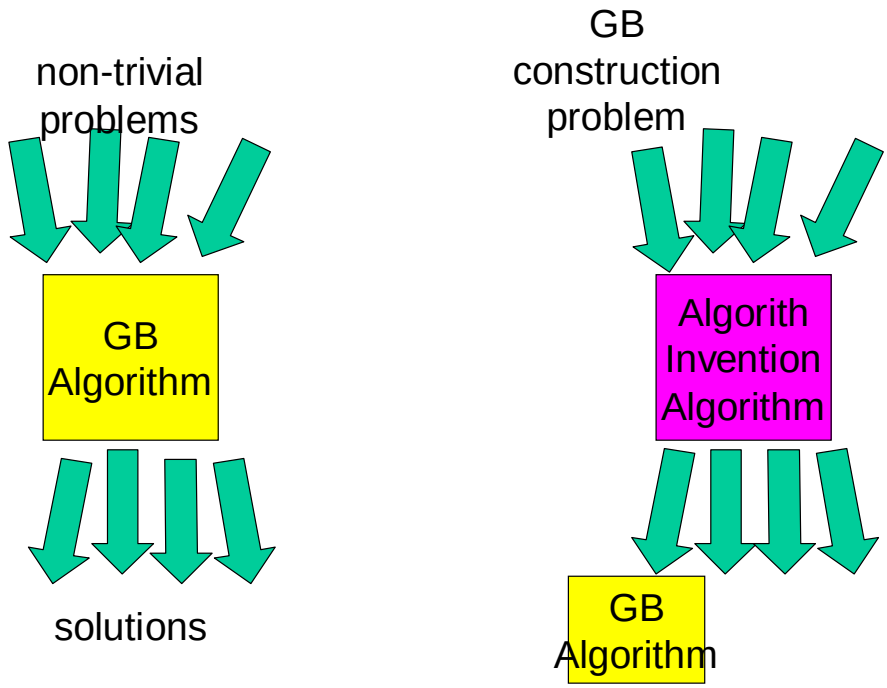


## Bad and Good for Me



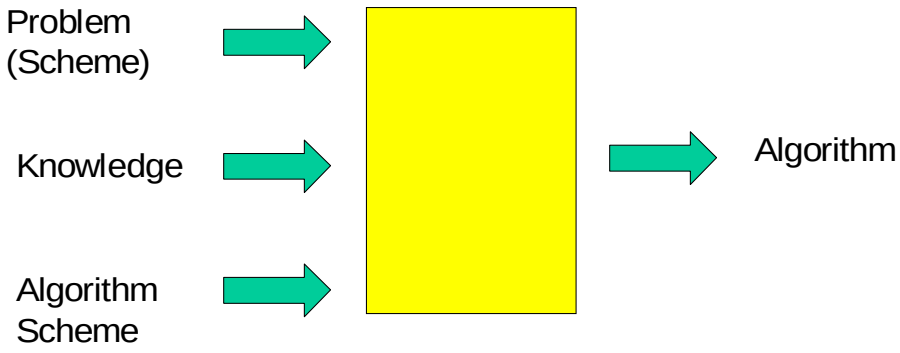

Good for me !

## Bad and Good for Me




54 of 55

## 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.

55 of 55

# 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, 2005, pp. 65-75.