Lazy Thinking: A New Method for Algorithm Synthesis

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

Goal

Algorithm Synthesis by Lazy Thinking

Example: Gröbner Bases

Goal

Algorithm Synthesis by Lazy Thinking

Example: Gröbner Bases

Computer Mathematics

- Numerics
- Computer Algebra
- Computational Logic
- ...

Unification, Interaction

Computational Algebra and Logic: BB., Editorial of J of Symbolic Computation 1985; Theorema 1995; ... "Calculemus":

Apply algebra to logic (e.g. Gröbner bases for geo theorem proving, ...)

Apply logic to algebra (e.g. automated verification and synthesis of algebraic algorithms, ...)

Computational Algebra and Numerics / Functional Analysis: SFB "Scientific Computing", Radon Institute RICAM, ...

E.g. numerical Gröbner bases (H. Stetter's book), ...

E.g. SFB 1322, Gröbner bases for operator theory, see talk by M. Rosenkranz

E.g. Special Semester on Gröbner bases at RICAM and RISC (Feb - July 2006):

Workshop "Approximate Commutative Algebra" (Feb 20-26, 2006)

The Theorema Project

The Theorema project at RISC aims at prototyping a system in wich the entire process of inventing and verifying mathematics ("mathematical theory exploration") should be supported:

- "automated proof generating" paradigm
- not "automated proof checking" paradigm.

A uniform logic and software frame: higher order predicate logic for both reasoning and computing.

Implementation language: Theorema.

Acknowledgement: T. Jebelean, W. Windsteiger, T. Kutsia, F. Piroi, M. Rosenkranz, and PhD students.

Some Reasoners in Theorema

- Predicate logic: natural deduction, S-decomposition
- Elementary analysis: PCS (alternating quantifiers)
- Set theory
- Induction on natural numbers, on tuples, ...
- Functors
- Equality, sequence variables
- Combinatorial identities
- Geometry (based on algebraic methods like Gröbner bases)
- "Lazy Thinking" method for algorithm synthesis
- ...

The PCS Method for Analysis Proving (BB 2001)

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

A Proof Generated by PCS

```
Definition["limit:", any[f, a],
    limit[f, a] ⟺ ∀_{ϵ>0} ∃_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]]
```

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

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

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

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

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

- ProofObject -

The following proof is generated completely automatically by this call to the PCS prover:

Prove:

(Proposition (limit of sum)) $\forall_{f,a,g,b}$ (limit[f, a] ∧ limit[g, b] ⇒ limit[$f + g$, $a + b$]),

under the assumptions:

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

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

and show

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

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

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

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

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

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

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

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

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

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

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

We assume

(8) $\epsilon_0 > 0$,

and show

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

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

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

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

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

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

which, using (4), is implied by:

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

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

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

Formula (14) is implied by

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

Partially solving it, formula (15) is implied by

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

Now,

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

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

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

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

Furthermore, we can immediately solve

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

for N_2^* by taking

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

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

□

Goal

Algorithm Synthesis by Lazy Thinking

Example: Gröbner Bases

The Algorithm Invention ("Synthesis") Problem

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

```
∀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 ...

Algorithm Synthesis by "Lazy Thinking" (BB 2002)

"Lazy Thinking" Method for Algorithm Synthesis =

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

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

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

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

Literature

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

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

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

> [Lau et al. 1999] K. K. Lau, M. Ornaghi, S. Tärnlund. Steadfast logic programs. Journal of Logic Programming, 38/3, 1999, pp. 259-294.
>
> And the work of A. Bundy and his group (U of Edinburgh) on the automated invention of induction schemes.

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

Problem: Synthesize "sorted" such that

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

("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

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

S, M, L, R are unknowns.

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

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

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):

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

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

$$\forall_{x}\ (L[x] \asymp R[x] \approx x)$$

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

What Do We Have Now?

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

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

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

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

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

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

Example: Synthesis of Insertion-Sort

Synthesize A such that

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

Algorithm Scheme: "simple recursion"

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

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

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

How Far Can We Go With the Method ?

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

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

- The problem was open for about 60 years.
- Dozens of (difficult) problems turned out to be reducible to the construction of Gröbner bases. (~ 600 papers, 10 textbooks, entry 13P10 in AMS index).

Goal

Algorithm Synthesis by Lazy Thinking

Example: Gröbner Bases

What are Gröbner Bases?

```
f1 = x y - 2 y z - z;
f2 = y^2 - x^2 z + x z;
f3 = z^2 - y^2 x + x;

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

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

```
{0.01 Second,
 {z + 4 z^3 - 17 z^4 + 3 z^5 - 45 z^6 + 60 z^7 - 29 z^8 + 124 z^9 - 48 z^10 + 64 z^11 - 64 z^12,
  -22001 z + 14361 y z + 16681 z^2 + 26380 z^3 + 226657 z^4 + 11085 z^5 -
   90346 z^6 - 472018 z^7 - 520424 z^8 - 139296 z^9 - 150784 z^10 + 490368 z^11,
  43083 y^2 - 11821 z + 267025 z^2 - 583085 z^3 + 663460 z^4 - 2288350 z^5 +
   2466820 z^6 - 3008257 z^7 + 4611948 z^8 - 2592304 z^9 + 2672704 z^10 - 1686848 z^11,
  43083 x - 118717 z + 69484 z^2 + 402334 z^3 + 409939 z^4 + 1202033 z^5 -
   2475608 z^6 + 354746 z^7 - 6049080 z^8 + 2269472 z^9 - 3106688 z^10 + 3442816 z^11}}
```

```
zsolexact1 = Solve[G[[1]] == 0, z]
```

```
{{z → 0},
 {z → Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 - 124 #1^8 + 48 #1^9 -
      64 #1^10 + 64 #1^11 &, 1]},
 {z → Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 2]},
 {z → Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 3]},
 {z → Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 4]},
 {z → Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 5]},
 {z → Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 6]},
 {z → Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 7]},
 {z → Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 8]},
 {z → Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 9]},
 {z → Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 10]},
 {z → Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 11]}}
```

One can compute with these "abstract roots" like with "square roots". For example

```
G0 = G /. zsolexact1[[2]] // Simplify
```

```
{0, Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 -
      60 #1^6 + 29 #1^7 - 124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]
   (-22001 + 14361 y + 16681 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 -
          60 #1^6 + 29 #1^7 - 124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1] +
      26380 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
            124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^2 +
      226657 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
            124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^3 +
```

```
    11085 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^4 -
    90346 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^5 -
    472018 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^6 -
    520424 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^7 -
    139296 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^8 -
    150784 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^9 +
    490368 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^10),
  -26357 + 43083 y^2 - 11821 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 -
      60 #1^6 + 29 #1^7 - 124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1] +
   161597 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^2 -
   135016 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^3 +
   584389 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^4 -
   1102285 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^5 +
   885400 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^6 -
   2243904 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^7 +
   1343680 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^8 -
   1327168 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^9 +
   985856 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^10,
  53794 + 43083 x - 118717 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 -
      60 #1^6 + 29 #1^7 - 124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1] +
   284660 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^2 -
   512164 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^3 +
   571321 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^4 -
   1218697 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^5 +
   752032 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^6 -
   1205280 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
          124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^7 +
```

```
621376 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^8 -
312640 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^9 +
336128 Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 - 60 #1^6 + 29 #1^7 -
      124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1]^10}
```

```
G0 /. {Root[-1 - 4 #1^2 + 17 #1^3 - 3 #1^4 + 45 #1^5 -
      60 #1^6 + 29 #1^7 - 124 #1^8 + 48 #1^9 - 64 #1^10 + 64 #1^11 &, 1] → α}
```

$$\{\alpha + 4\alpha^3 - 17\alpha^4 + 3\alpha^5 - 45\alpha^6 + 60\alpha^7 - 29\alpha^8 + 124\alpha^9 - 48\alpha^{10} + 64\alpha^{11} - 64\alpha^{12},$$
$$-22001\alpha + 14361\, y\,\alpha + 16681\alpha^2 + 26380\alpha^3 + 226657\alpha^4 + 11085\alpha^5 - 90346\alpha^6 - 472018\alpha^7 - 520424\alpha^8 - 139296\alpha^9 - 150784\alpha^{10} + 490368\alpha^{11},$$
$$43083\, y^2 - 11821\alpha + 267025\alpha^2 - 583085\alpha^3 + 663460\alpha^4 - 2288350\alpha^5 + 2466820\alpha^6 - 3008257\alpha^7 + 4611948\alpha^8 - 2592304\alpha^9 + 2672704\alpha^{10} - 1686848\alpha^{11},$$
$$43083\, x - 118717\alpha + 69484\alpha^2 + 402334\alpha^3 + 409939\alpha^4 + 1202033\alpha^5 - 2475608\alpha^6 + 354746\alpha^7 - 6049080\alpha^8 + 2269472\alpha^9 - 3106688\alpha^{10} + 3442816\alpha^{11}\}$$

No more details in this talk!

Alternatively, compute numerically:

```
zsol = NSolve[G[[1]] == 0, z]
```

```
{{z → -0.331304 - 0.586934 i}, {z → -0.331304 + 0.586934 i},
 {z → -0.296413 - 0.705329 i}, {z → -0.296413 + 0.705329 i},
 {z → -0.163124 - 0.37694 i}, {z → -0.163124 + 0.37694 i},
 {z → 0.}, {z → 0.0248919 - 0.89178 i}, {z → 0.0248919 + 0.89178 i},
 {z → 0.468852}, {z → 0.670231}, {z → 1.39282}}
```

```
Gsubnum = G /. zsol[[1]]
```

$$\{1.33227\times 10^{-15} + 9.71445\times 10^{-17}\, i,$$
$$(-523.519 - 4967.65\, i) - (4757.86 + 8428.97\, i)\, y,$$
$$(-7846.9 - 8372.06\, i) + 43083\, y^2,\ (-16311.7 + 16611.\, i) + 43083\, x\}$$

```
PolynomialGCD[Gsubnum[[2]], Gsubnum[[3]]]
```

```
1
```

```
ysol = NSolve[ Gsubnum[[2]] == 0, y]
```

```
{{y → -0.473535 - 0.205184 i}}
```

```
ysol = NSolve[ Gsubnum[[3]] == 0, y]
```

```
{{y → -0.473535 - 0.205184 i}, {y → 0.473535 + 0.205184 i}}
```

Theorem (Roider, Kalkbrener et al. 1990): It suffices to consider the poly in y with lowest degree.

```
xsol = NSolve[ Gsubnum[[4]] == 0, x]
```

$\{\{x \to 0.378611 - 0.385558\, i\}\}$

```
F /. zsol[[1]] /. ysol[[1]] /. xsol[[1]]
```

$\{-1.88738 \times 10^{-15} + 2.35922 \times 10^{-16}\, i,$
$1.02696 \times 10^{-15} + 1.52656 \times 10^{-16}\, i,\ -1.94289 \times 10^{-15} - 1.11022 \times 10^{-16}\, i\}$

Definition of Groebner Bases

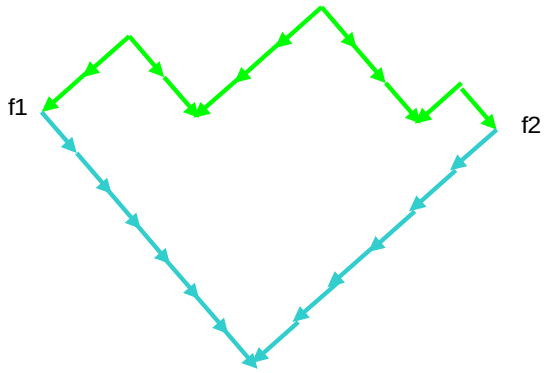
A set F of polynomials is called a Groebner basis (w.r.t. a chosen ordering of power products) iff

$\to_F$ is confluent.

$\to_F$ ... "a division step".

Confluence of Division $\to_F$

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



25 of 47

The "Main Theorem" of Gröbner Bases Theory (BB 1965):

F is a Gröbner basis $\iff \forall_{f_1, f_2 \in F}$ remainder[F, S–polynomial[f_1, f_2]] = 0.

Proof: Nontrivial. Combinatorial. Some details see below.

The theorem reduces an infinite check to a finite check: Recall definition of "F is a Gröbner basis":

for infinitely many f_1, f_2 one has to ...

Remark: If F is a Gröbner basis then it has many other other useful properties (e.g. "elimination property").

The power of the Gröbner bases method is contained in this theorem and its proof.

S-Polynomials: The Main Notion of Algorithmic GB Theory

```
f1 = -2 y + x y
f2 = -x^2 + y^2
```

$-2\,y + x\,y$

$-x^2 + y^2$

```
S-polynomial[f1, f2] = y f1 - x f2
```

$y\,(-2\,y + x\,y) - x\,(-x^2 + y^2)$

```
S-polynomial[f1, f2] = y f1 - x f2    // Expand
```

$x^3 - 2\,y^2$

The Problem of *Constructing* Gröbner Bases

Given F, find G s.t. Ideal(F) = Ideal(G) and G is a Gröbner basis.

(Ideal(F) := the set of all linear combinations $h_1 f_1 + \ldots + h_m f_m$

with $f_1, \ldots, f_m \in$ F and $h_1, \ldots, h_m$ arbitrary polynomials.)

An Algorithm for *Constructing* Gröbner Bases (BB 1965)

Recall the main theorem:

$$F \text{ is a Gröbner basis} \iff \forall_{f_1, f_2 \in F} \text{ remainder}[\, F, \text{S–polynomial}[f_1, f_2]] = 0.$$

Based on the main theorem, the problem can be solved by the following algorithm:

Start with G:= F.

For any pair of polynomials $f_1, f_2 \in G$:

h := remainder[G, S–polynomial[f_1, f_2]]

If $h = 0$, consider the next pair.

If $h \neq 0$, add h to G and iterate.

Correctness and Termination of the Algorithm

Correctness: Easy as soon as we know the main theorem.

Termination: by Dickson's Lemma (Dickson 1913, BB 1970).

Specializations

The Gröbner bases algorithm,

for linear polynomials, specializes to Gauss' algorithm, and

for univariate polynomials, specializes to Euclid's algorithm.

Example

Let's again look at

```
f1 = -2 y + x y
f2 = -x^2 + y^2
```

$-2\,y + x\,y$

$-x^2 + y^2$

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

$\{-2\,y + x\,y,\ -x^2 + y^2\}$

F is not a Groebner basis.

The S-polynomial of f_1, f_2:

```
S-polynomial[f1, f2] = y f1 - x f2    // Expand
```

$x^3 - 2\,y^2$

Its remainder w.r.t. F is:

```
-2 x^2 + x^3.
```

All the other S-polynomials have remainder 0. Hence, we arrived at a Groebner basis.

The Groebner basis algorithm is available now available in all math software systems, e.g. in *Mathematica*:

```
G = GroebnerBasis[F, {y, x}]
```

$\{-2x^2 + x^3, -2y + xy, -x^2 + y^2\}$

32 of 47

The Meta-Problem of Automatically Finding an Algorithm for Constructing Gröbner Bases: Apply "Lazy Thinking"

Find algorithm Gb such that

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

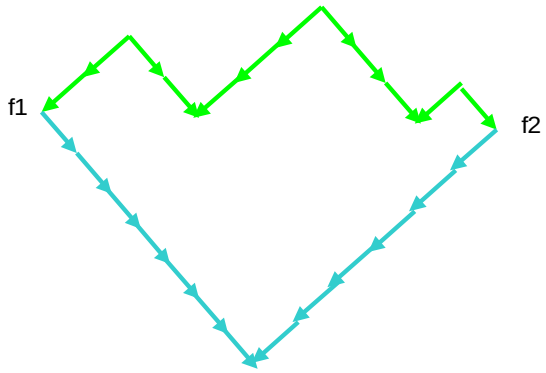
```
is-Gröbner-basis[G] ⇔ is-confluent[ →G ].
```

$\to_G$ ... a division step.

33 of XXX

Recall: Confluence of Division $\to_G$

```
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, ⟨⟩] = F

A[F, ⟨⟨g1, g2⟩, p⟩] =
 where[f = lc[g1, g2], h1 = trd[rd[f, g1], F], h2 = trd[rd[f, g2], F],
  { A[F, ⟨p⟩]                                                  ⇐  h1 = h2
  { A[F ~ df[h1, h2], ⟨p⟩ ≈ ⟨⟨F_k, df[h1, h2]⟩ |_{k=1,…,|F|}⟩]  ⇐  otherwise ]
```

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

```
∀_{f,g} (f ≥ 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

```
∀_F ( is-finite[ A[F] ]
      is-Gröbner-basis[ A[F]]
      ideal[F] = ideal[ A[F]]. )
```

using the above scheme for A we can *automatically* produce the idea of S-polynomial, i.e. the idea that

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

and

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

are suitable choices in the algorithm scheme.

Now Start the (Automated) Correctness Proof

With current theorem proving technology, in the *Theorema* system (and other provers), the proof attempt can be done automatically.

Ongoing PhD thesis by A. Craciun.

Details

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

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

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

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

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

Using Available Knowledge

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

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

Newman's lemma (1942):

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

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

$$f1 \downarrow^* f2 \Leftrightarrow \exists_g \left(\begin{cases} f1 \to^* g \\ f2 \to^* g \end{cases}\right)$$

The (Automated) Proof Attempt

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

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

We have to find a polyonomial g such that

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

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

$$lp[g1] \mid p,$$
$$f1 = rd[p, g1],$$
$$lp[g2] \mid p,$$
$$f2 = rd[p, g2].$$

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

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

where

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

Case h1=h2

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

(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,

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

Now we are stuck in the proof.

42 of 47

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

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

With such an lc, we then would have

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

and, hence,

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

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

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

43 of 47

Summarize the (Automatically Generated) Specifications of the Subalgorithm lc

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

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

and the requirements:

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

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

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

```
∀h1,h2 (h1 ↓{df[h1,h2]}* h2)  (df requirement).
```

46 of 47

Looking to the Knowledge Base for a Suitable df

47 of 47

Conclusion

- Theorem proving methods, together with suitable strategies for systematic mathematical theory exploration, are reaching a level of sophistication where the automated invention of correct non-trivial algorithms seems to be possible.
- This is just one aspect of the overall goal of formal mathematics: Create an environment of reasoning tools for building up large, provenly correct, well structured, formal mathematical knowledge bases (concepts, propositions, problems, algorithms).
- "(Anti-) Bourbakism of the 21st Century".

References

Appendix: The Proof Notebooks Automatically Generated by Theorema for the Synthesis of the Merge-Sort Algorithm