Algorithmic Algorithm Synthesis:

Case Study Gröbner Bases

Bruno Buchberger

RISC, Austria

Dedicated to Bob F. Caviness

Outline:

The Theorema Project

An "Algorithm" for Algorithm Synthesis

Synthesis of a Gröbner Bases Algorithm

Conclusion

The Theorema Project

An Algorithm for Algorithm Synthesis

Algorithmic Synthesis of a Gröbner Bases Algorithm

Conclusion

Current Mathematical Knowledge

- algorithm libraries, e.g. *Mathematica*, Maple, etc.
- decision algorithms and proof generators for certain logical theories
- proof checkers and proof generators for predicate logic
- ....
- mathematical "knowledge" in paper form, LaTeX files etc.

What "We" Want

Integrated Systems

Integrated systems that support (partially automate) the entire "mathematical theory exploration" process:

Invention and Verification

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 new knowledge (definitions, propositions, problems, algorithms) in structured knowledge libraries so that they can be used easily in the next rounds of mathematical theory exploration.

- **For example:**

start from: +, -, *, ..< on reals (axioms, propositions, problems, algorithms)

new concepts: definitions of "sequence" and operations +, -, ... on sequences
explore: propositions, problems, algorithms on sequences

new concepts: "limit", "continuity"
explore: propositions, problems, algorithms on sequences

- **A benchmark example:**

Computer-supported development of the theory of Groebner bases and related theories.

6 of XXX

The *Theorema* Project

The *Theorema* project [BB et al. 1996 ...] aims at being a frame for supporting mathematical theory exploration in the above sense.

The *Theorema* language is a version of predicate logic.

A sublanguage of the *Theorema* language is a programming language.

The *Theorema* system is implemented in Mathematica. (This may change in the future.)

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

7 of XXX

Mathematical Knowledge Management: A Recent International Endeavor

The *Theorema* group is (an initiator and member) of the international MKM (Mathematical Knowledge Management) Network (NuPrl, Izabelle, MIZAR, ...)

Numerical Mathematics

Computer Algebra

Automated Theorem Proving

Mathematical Knowledge Management
("Symbolic Computation" in its widest sense)

8 of XXX

✓ The Theorema Project

An "Algorithm" for Algorithm Synthesis

Algorithmic Synthesis of a Gröbner Bases Algorithm

Conclusion

9 of XXX

■ The Algorithm Invention ("Synthesis") Problem

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

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

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 (cf. B. Caviness 1970, ...) but ...

There is a rich literature on algorithm synthesis methods.

10 of 10

■ The "Lazy Thinking" Method (BB 2001): A Very Rough Sketch

Given: a problem specification P.

For finding an algorithm A that satisfies $\underset{x}{\forall}\ \mathbf{P}[x, \mathbf{A}[x]]$,

- we generate (automatically) a couple of (hopefully) simpler problems Q, R, ...,

- we synthesize algorithms B, C, ... for Q, R, ...

- from B, C, ... we compose (automatically) an algorithm A.

When can we terminate?
When we arrive at problems, for which algorithms are already known.

11 of XXX

■ The "Lazy Thinking" Method: More Details

Given a problem specification P

- consider various "algorithm schemes" for A
- and try to prove (automatically) $\underset{x}{\forall}\ \mathbf{P}[x, \mathbf{A}[x]]$.
- This proof will normally fail because nothing is known on the unspecified sub-algorithms in the algorithm scheme.
- From the temporary assumptions and goals in the failing proof situation (automatically) generate such specifications for the unspecified sub-algorithms that would make the proof possible.

Now, apply the method recursively to the auxiliary functions.

This is "lazy"

- we use the condensed algorithmic experience of others (in the schemes)
- we start (routine) proving before we have an algorithm
- we just wait until we fail in order to get an idea from the failure.

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

Problem

Synthesize "sorted" such that

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

("Correctness Theorem")

Knowledge on Problem

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

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

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

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

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

etc.

An Algorithm Scheme

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

("divide and conquer")

We Now Start Proving the Correctness Theorem and Analyze the Failing Proof

The Result of Applying Lazy Thinking

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

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

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

$$\underset{x}{\forall}\ (\text{left-split}[x] \asymp \text{right-split}[x] \approx x)$$

What Do We Have Now?

- **Case A:** We find algorithms special, merged, left–split, right–split in our knowledge base for which the properties specified above are already contained in the knowledge base or can be derived from the knowledge base.
  In this case, we are done, i.e. we have synthesized a sorting algorithm.

- **Case B:** We do not find algorithms special, merged, left–split, right–split in our knowledge base for which the properties specified can be proved.
  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).

How Can we Teach (= Automate) the Method

- Compile a library of "algorithm design experience" (= algorithm schemes).
- Teach (automate) proving.
- Teach (automate) generation of useful specifications of sub-algorithms from failing correctness proofs:

A simple (but amazingly powerful) rule:

Collect temporary assumptions T[x0, ... A [ ], ...]
and temporary goals G[x0, ...m [A [ ]]]

and produces specification

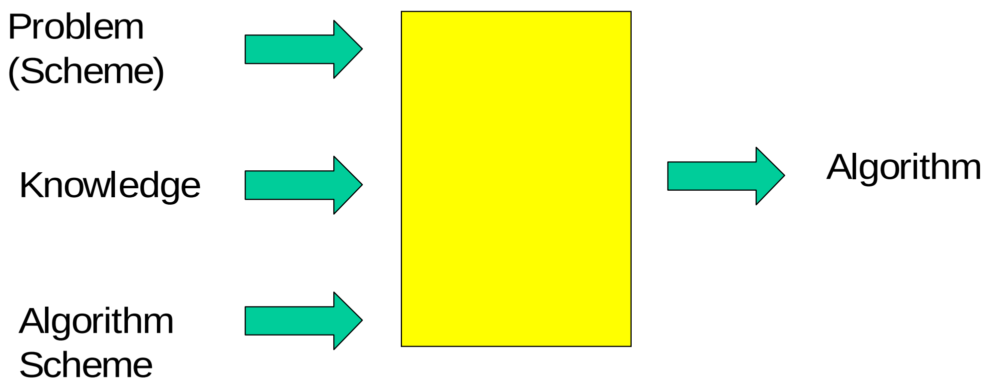
$$\underset{x, \ldots, Y, \ldots}{\forall} \left(T[X, \ \ldots Y, \ \ldots] \Rightarrow G[Y, \ \ldots m \ [Y]] \right).$$

Details: see papers [BB 2003] and example.

Research topic: other rules.

14 of XXX

Parameters for Lazy Thinking



Lazy Thinking

15 of XXX

Example: Synthesis of Insertion-Sort

Problem

Synthesize A such that

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

Algorithm Scheme: "simple recursion"

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

Resulting Specification for Subalgorithms

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

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

Details of an Automated Synthesis

See the notebooks automatically produced by *Theorema* for the insertion-sort example.

16 of XXX

How far can we go with this method?

Can we automatically synthesize algorithms for non-trivial problems?

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

What is "non-trivial"?

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

Hence, question: Can Lazy Thinking automatically invent the notion of S-polynomial and automatically deliver the S-polynomial theorem.

17 of XXX

✓ What we Have, What we Want

✓ An Algorithm for Algorithm Synthesis

Algorithmic Synthesis of a Gröbner Bases Algorithm

Conclusion

The Problem of Constructing Gröbner Bases

Find algorithm Gb such that

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

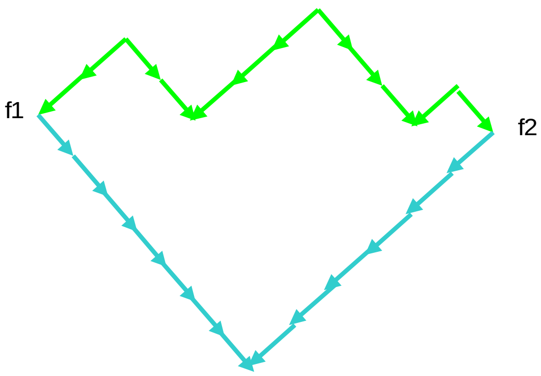
Definitions [BB 1965, 1970]:

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

Confluence of Division $\rightarrow_G$

$$(h1\rightarrow_G h2)\Leftrightarrow\underset{g\in G}{\exists}\left(\left\{\begin{array}{l}\texttt{lp}[g]\mid\texttt{lp}[h1]\\ h2=h1-(\texttt{lm}[h1]/\texttt{lm}[g])\,g\end{array}\right.\right),$$

$$\textbf{is-confluent}[\to] \;:\Leftrightarrow\; \underset{f1,f2}{\forall}\left(f1 \leftrightarrow^{*} f2 \Rightarrow f1 \downarrow^{*} f2\right)$$



20 of XXX

Knowledge on the Concepts Involved

$$h1 \to_{G} h2 \;\Rightarrow\; p\,.\,h1 \to_{G} p\,.\,h2$$

etc.

21 of XXX

Algorithm Scheme "Critical Pair / Completion"

$$\mathbf{A}[F] = \mathbf{A}[F, \mathrm{pairs}[F]]$$
$$\mathbf{A}[F, \langle\rangle] = F$$

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

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:

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

22 of XXX

The Essential Problem

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

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

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.

23 of XXX

Now Start the (Automated) Correctness Proof

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

24 of XXX

Details

Upon Termination

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

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

- **We Have to Prove**

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

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

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

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

- **Assumption**

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

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

We have to find a polyonomial g such that

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

- **From the Assumption**

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

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

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

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

where

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

- **Case h1=h2: In this case**

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

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

Hence, by elementary properties of polynomial reduction,

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

Now we are stuck in the proof.

- **Use Specification Generation Algorithm**

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

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

With such an lc, we then would have

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

and, hence,

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

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

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

Summarizing the Specifications of the Unknown Subalgorithm lc

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

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

and the requirements:

$$\text{rd[lc[g1, g2], g1]} \prec \text{lc[g1, g2]},$$
$$\text{rd[lc[g1, g2], g2]} \prec \text{lc[g1, g2]},$$

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

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

A Suitable lc

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

is a suitable function that satisfies the above requirements.

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

Case h1≠h2 and, hence, df[h1,h2]∈G:

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

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

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

(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

$$\mathbf{df}[\mathbf{h1}, \mathbf{h2}] = \mathbf{h1} - \mathbf{h2},$$

because, in fact,

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

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

Namely,

Summary of the Synthesizing Proof Attempt

Failure Situation

The proof, of course, fails at the point where it would need knowledge about the unknown subalgorithms lc and df.

Beginning of the proof:

Let G be A[F]. We have to prove that

$$\begin{aligned} &\textbf{is-finite}[\mathbf{G}], \\ &\textbf{ideal}[\mathbf{F}] = \textbf{ideal}[\mathbf{G}], \\ &\textbf{is-Gröbner-basis}[\mathbf{G}], \\ &\qquad \textbf{i.e. is-confluent}[\rightarrow_{G}]. \end{aligned}$$

Assumption

We only deal with the third, most important, property. For this, we assume

$$\begin{cases} p \rightarrow_{G} f1 \\ p \rightarrow_{G} f2. \end{cases}$$

and have to find a polynomial g such that

$$\begin{aligned} &f1 \rightarrow_{G}{}^{*} g, \\ &f2 \rightarrow_{G}{}^{*} g. \end{aligned}$$

Generation of the Specification of lc

In the failing proof situation, by the (automated) analysis algorithm sketched above, we detect that the proof could be completed if the unknown **lc** satisfied the following property:

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

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

Eureka! It is clear that this specification is (only) met by

$$\mathtt{lc}[g1, g2] = \mathtt{lcm}[\mathtt{lp}[g1], \mathtt{lp}[g2]].$$

27 of XXX

Generation of the Specification of df

Similarly, it can be (automatically) detected that

$$\mathtt{df}[h1, h2] = h1 - h2.$$

28 of XXX

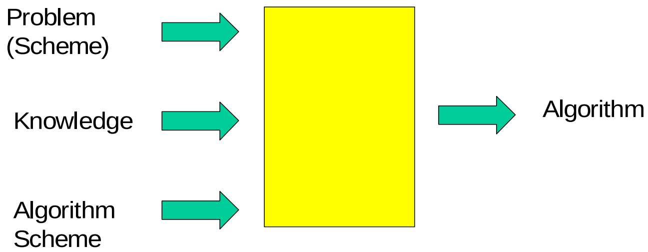
✓ What we Have, What we Want

✓ An Algorithm for Algorithm Synthesis

✓ Algorithmic Synthesis of a Gröbner Bases Algorithm

Conclusion

■ Research Topics



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

  More generally, libraries of definition, theorem, problem, and algorithm schemes.

- **Case studies** of problem (schemes), knowledge, algorithm schemes and how they produce algorithms.
- Improved **algorithms for generating problem specifications** from failing proofs.
- "F**unctor" knowledge:** each algorithm is a "functor"

  If s, m, l, r is a merge structure then A is a sort structure.
  and A results from s, m, l, r by divide and conquer

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.