... Vibe Mathematics

Bruno Buchberger

RISC

Johannes Kepler University

2026 07 10

I am excited:

- Machine Learning (ML),
- a „numerical method“ (essentially an „interpolation“ method),
- an unexpected approach to algorithms for algebraic problems,
- an unexpected („vibe“) approach to automated mathematical invention,

I am excited:

- an unexpected central role for automated proof checking (and generation? THEOREMA!),
- Symbolic Computation (SC) has an (the) important role for the future of AI,
- high motivation for all involved in SC, … RISC, JSC, SCM, RISC European College, …

- My talk is obsolete: see LeanMistral, Aristotle, Cezary‘s Talk, …
- Except, maybe, one point …

Content

- Inventing mathematics; in particular, algorithms
- Learning algorithms for algebraic problems
- Inventing and proving algorithms (theories) by „LLMs + AR“ (= „Vibe Coding“, „Vibe Proving“, „Vibe Mathematics“)
- An Example: A simple problem with a non-trivial proof
- My Personal Research Plan

Inventing mathematics; in particular, algorithms

Usual approach for finding algorithms:

Given: „complete“ problem specification P. („Given x, find y s.t. P(x,y).“)

Find: algorithm A, s.t. „for all x, P(x, A(x))“.

„Think“.

Propose A and prove “for all x, P(x, A(x))“.

More generally, usual approach for inventing new theories:

Given: a couple of theories (formulae):

definitions (e.g. addition), problems, theorems, algorithms, proofs.

Find (with some „motivation“ in mind): a new theory (formulae):

definitions (e.g. multiplication), problems, theorems, algorithms, proofs.

„Think“.

Propose and prove, step by step, new definitions, problems, …

(„Generate and verify“; „hot phase and cool phase of mathematics“, …)

The mathematical invention process itself can be seen as a mathematical problem:

Given: certain formulae …

Find: other formulae that satisfy certain properties

(e.g. that a certain algorithms solve certain problems)

Can we automate this process?

(This is of highest relevance because mathematics is the basis of all automation.

The automation of mathematics, hence, is the „eye in the automation hurrican“)

The mathematical invention process itself can be seen as a mathematical problem:

Mathematics applied to itself, „meta-mathematics“.

Mathematics is essentially „meta“ since it‘s beginnings:

„Think once deeply and you need not think anymore infinitely many times …“

Meta-mathematics is very concrete:

E.g. Gröbner bases theory applied to geo theorem proving.

(Proved theories become provers, „Reflexion“, …, see BB book „Science and Meditation“)

The mathematical invention process itself can be seen as a mathematical problem:

1920 +-: Hilbert‘s program and Gödel‘s (In-) Completeness Theorem

The (positive!) motivation for automating math invention for all times to come …

The mathematical invention process itself can be seen as a mathematical problem:

1940 +-: Bourbakism:

All of math in one logical frame (first order set theory).

However, the notion of „algorithm“ does not appear in any of the Bourbaki volumes!

Thus, also, „algorithmic“ math invention was out of question.

The mathematical invention process itself can be seen as a mathematical problem:

2014 +- (International Math Congress): The „Mathematical Knowledge Management“ Movement (MKM)

BB, Kohlhase et al.

Goal: A universal „platform“ for the invention of math.

Had little impact because the „Generate“ part of invention had little algorithmic support.

(In particular, automated „proof checking“ did not seem to be attractive without automated proof generation!)

However, strong impulses in this direction:

- Wolfram Alpha

- The LEAN / Isabelle / …. community

- The Hoskinson Center for Formal Mathematics at Carnegie Mellon Univ. 2021, directed by Jeremy Avigad, sponsored by Charles Hoskinson (founder of Cardano, cryptocurrency),

  20 Mio $

- Still not “the end of the story”

The mathematical invention process itself can be seen as a mathematical problem:

2022 - …: An (unexpected) new kick by ML (in particular also, LLMs).

Approach A:

„Learn“ (correct?) algorithms for algebraic problems from known input / output pairs

Approach B:

Generate proposals for algorithms (and entire theories) by „LLMs“.

Verify the correctness by Automated Reasoning (in particular, proof checkers).

Approach A: ML supports SC

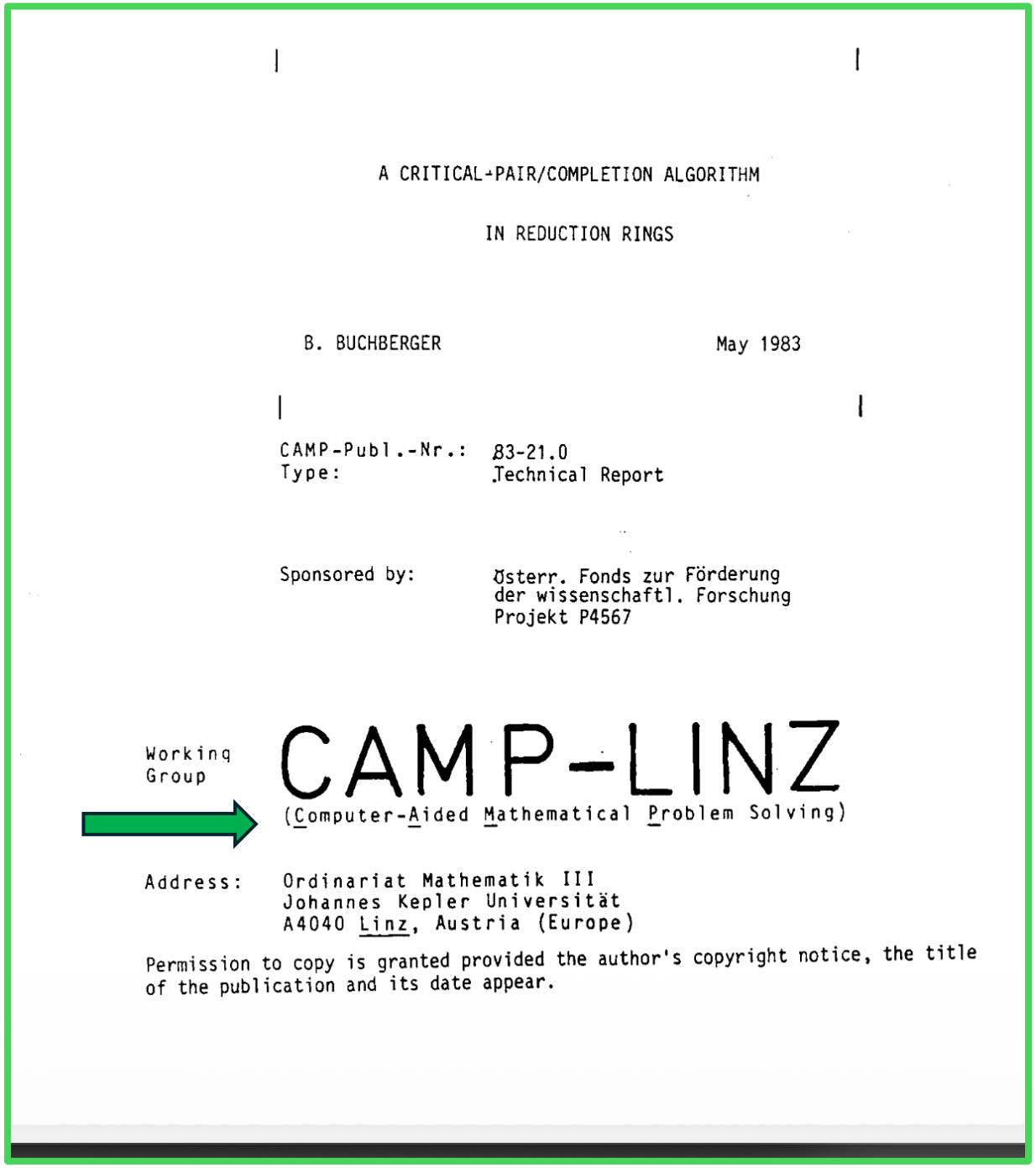
Approach B: SC supports ML

The mathematical invention process itself

can be seen as a mathematical problem:

Personally, this was my view on mathematics and

my „political goal“ for my working group since 1982:



A CRITICAL-PAIR/COMPLETION ALGORITHM

IN REDUCTION RINGS

B. BUCHBERGER

May 1983

CAMP-Publ.-Nr.: 83-21.0
Type: Technical Report

Sponsored by: Österr. Fonds zur Förderung
der wissenschaftl. Forschung
Projekt P4567

Working Group

CAMP-LINZ

(Computer-Aided Mathematical Problem Solving)

Address: Ordinariat Mathematik III
Johannes Kepler Universität
A4040 Linz, Austria (Europe)

Approach A:
Learning algorithms for algebraic problems

Examples of algebraic problems („completely specified“ problems P(x,y)):

long integer addition,

polynomial factorization,

Gröbner bases construction,

cylindrical algebraic decomposition,

symbolic integration,

symbolic solution of differential and difference equations, (SCDDE 2026 …),

….

„Bold“ approach:

- Given a problem P.

- Take (many) known input / output pairs satisfying $P(x_1,y_1), \ldots, P(x_n,y_n)$

- and „train“ a neural network (NN),

  i.e. determine the unknown „weights“ $(w_1,\ldots,w_k)$ in a simple algorithm skeleton A

- s.t. the trained A behaves correctly on the inputs $x_1, \ldots, x_n$, and „hopefully“ also on “many“ others.

Why should we do that?

? We already know an exact algorithm for the problem that works provenly correct for all possible inputs!

? The trained network does not at all reflect any mathematical ideas on which the problem definition and exakt algorithms are based!

? A finite NN (which, essentially, is a nested if-then-else expression) cannot express a non-trivial relation defined by recursion! (The proof of the „universality“ of NN refers to the representability of continuous functions and not to the „Turing-completeness“ of NN. ...??)

+ However, the trained network could perhaps compute much faster than the known exact algorithms! And, in fact, this is the experience!

The bold approach was pursued by courageous people: Kera et al.

- E.g. for Gröbner bases computation, …
- How can one get „sufficiently big“ training sets: Solved by going from GB back to poly sets.
- Encouraging: dramatic speed-up!
- Question: Can one give an exact description of the domain, in wich the trained NN will guarantee exact results?
- How can one verify the exactness of the results? E.g.: Is the output a Gröbner basis? Does the output generate the same ideal as the input? How costly are these checks?

Approach B:

Inventing and proving algorithms (theories) by „LLMs + AR“

(= „Vibe Coding“, „Vibe Proving“, „Vibe Mathematics“)

Given: certain formulae …

Find: other formulae that satisfy certain properties

(e.g. that a certain algorithms solve certain problems)

Approach B.1: Invent symbolic (correct) algorithms (rules) for invention steps

Approach B.2: Ask an LLM (“Math invention can be learned from the math literature!“)

Approach B.3: Send a plan, desire, … for a new theory as a prompt to an „LLM“ and

check the results with a proof checker (if „sorry“, repeat in a “cylcle“)

Approach B.1: “Logical Invention Rules“

This was and is the approach of the THEOREMA system (BB, WW et al.)

Example of a logical „Rule“ (Algorithm on the meta-level): „Lazy Thinking“

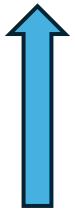
- Start with a proof (e.g. a correctness proof for an algorithm) with an „algorithm schema“ and when the proof fails, analyze the proof and specify subalgorithms whose correctness will enable the continuation of the proof.
- Worked well (e.g. automated re-invention of my GB algorithm).
- I had to give up work on this approach due to my many managment duties (and little attention from community)

Sketch of an example:

Definition: lim(f, a) iff for all e > 0 ex. N s.t. for all n > N abs (f (n) - a) < e.

Lemma: If lim(f, a) and lim(g, b) then lim(f+g, a+b).

Proof: … for all n > max(N_f (e/2), N_g(e/2) abs((f+g)(n) - (a+b)) < e) …



such „solving terms“ (algorithms) are found automatically in THEOREMA

Approach B.2: „Ask the Literature by LLM“

- Essentially, this is just „looking up and summarizing“ the mathematical literature on the problem, e.g. the proof of a proposition, by the „LLM type of ML“.
- Worked amazingly well in the early days of LLM.

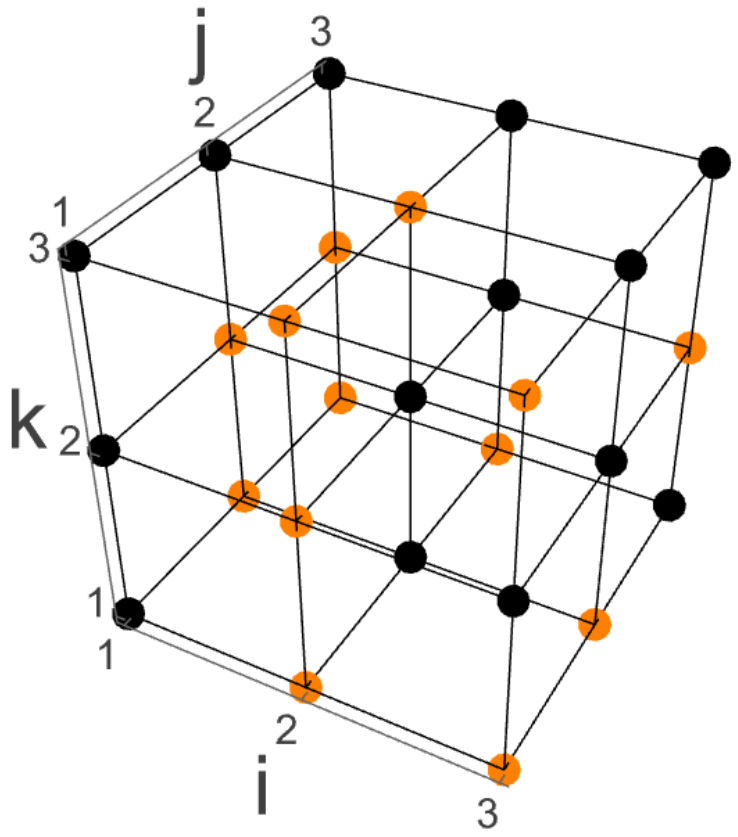
- However, open to (intolerable) „halluzinations“ - like everything in LLMs.
- Recently, LLMs get much better for mathematical invention because B.3 may be “under the hood“ of an LLM. (i.e. the job is not done by ML, but by ML + proof checking)

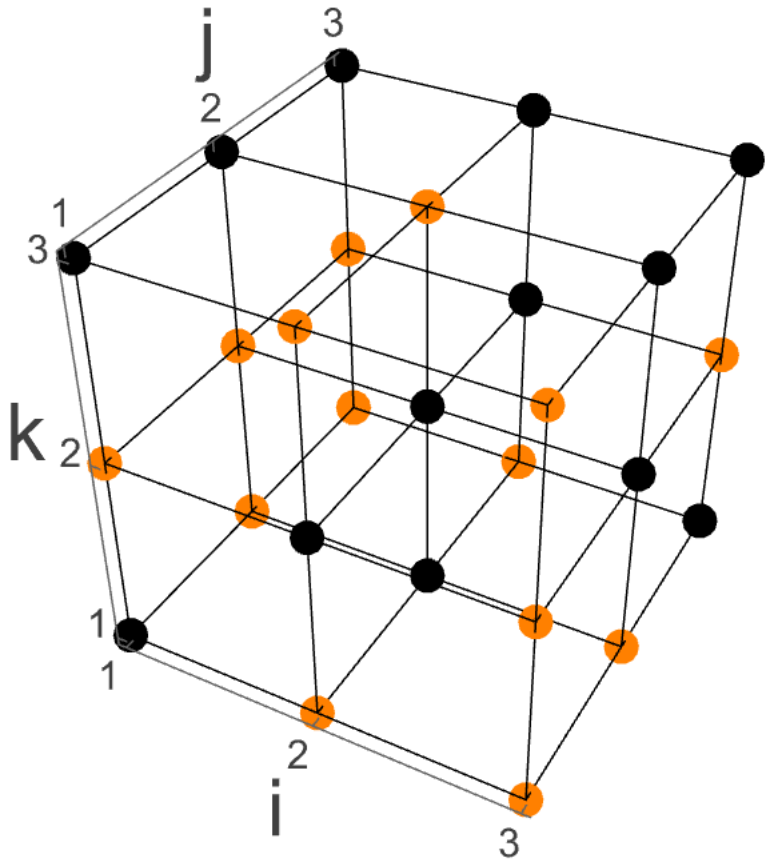
Approch B.3.: „LLM + Proof Checker‘

- This approach is becoming very popular now with impressive results, e.g. „Aristotle“ by Harmonics. (300 Mio $ VC.), LeanMistral, …
- Institute for Computer-Aided Reasoning in Mathematics, Carnegie Mellong, directed by Jeremy Avigad.
- It will change the way we „do mathematis“.
- However, not so impressive in situations with little background in the published literature, see the example.

An Example: A simple problem with a non-trivial proof

A „touch“

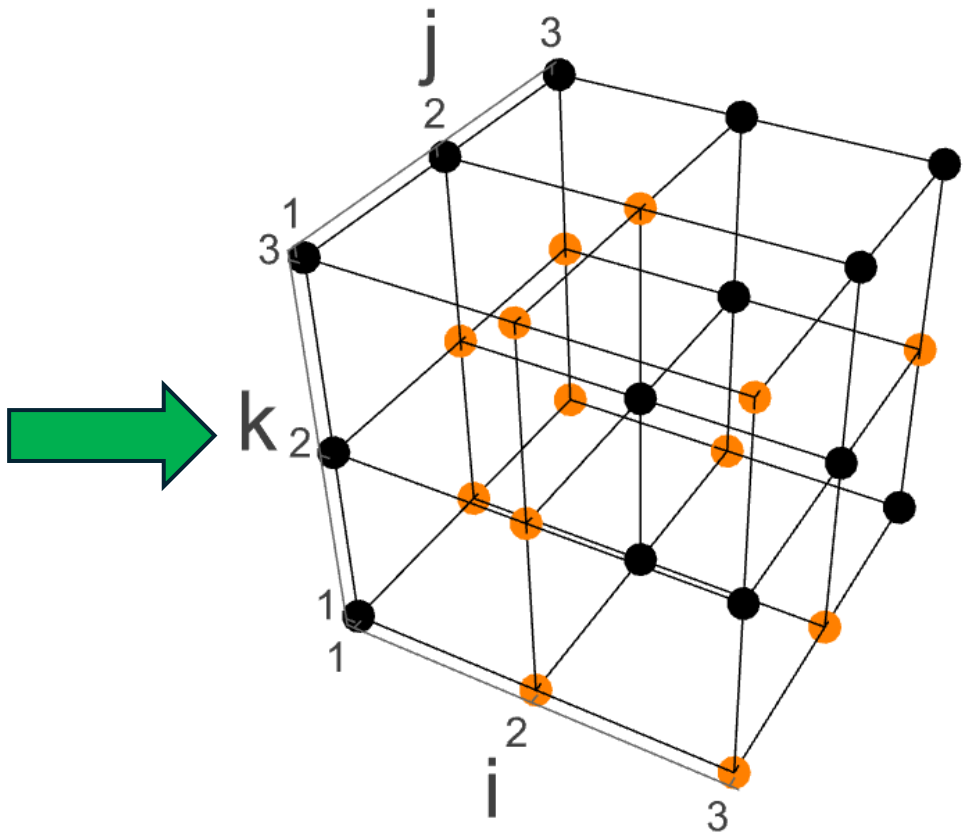




Problem:

Can one get to „all off“

by a finite sequence of touches?



The Experiment (since Dec 2025):

- I developed a proof for a (seemlingly too simple, but surprisingly correct) algorithm „by hand“ and observed myself in this process.

  (See BB, RISC Tech Report 2016-05)

- Koji and I, in parallel, experimented with various versions of combining LLMs and Automated Reasoning Systems (like LEAN) in order to get a proof for this algorithm (with and without my proof as part of the prompt).

  (See our joined paper at this conference.)

Result:

- Some LLMs quite disappointing.

- Github Copilot performed by far best:
  - found all three key ideas (lemmas) for the proof,
  - managed complete verification by LEAN,
  - composed nice „readable“ version of the proof
  - only small formal deficiencies in the informal presentation
  - only slight deviation from the actual proof goal

- The only remaining strength of my proof over the Copilot proof:

  I had the „proof scheme“ of „canonical simplification“ in my tool belt!

Summary of the experience from the experiment:

I am deeply impressed

and motivated.

Research Plan

I want to resume work on the development of THEOREMA because

- its focus was on formal tools („formulae schemes“) for the generation of theories (including proofs) without LLMs
- and I hypothesize that

“formal theory generation + math literature LLMs“

should be more powerful than

„formal theory checking + math literature LLMs“

What is the difference for getting ideas for a new piece of mathematics?

- Formal rules like „Lazy Thinking“ or „Schemes“:
  - can be run locally with no costs
  - can create new ideas (not in the literature)
  - reflect the creative ideas of the system (rules …) designer

- Ideas found by LLMs from the literature:
  - need enormous search effort
  - repeat what is known
  - collect the published ideas

Future systems for the (semi-)automation of the mathematical invention process should incorporate:

- tools for graphical experimentation
- computation inside the formal reasoning system for computatonal experimentation
- interaction with mathematical algorithm libraries and systems like Mathematica
- Interaction with "LLM" type mathematical knowledge search
- proof checking
- *formal theory generation (definitions, problems, theorems, algorithms, proofs) by formal tools like "formula schemes" and „lazy thinking"*
- *build-up and management of mathematical knowledge bases (the „functor principle")*
- *reasoning about the complexity of algorithms inside the same system*
- *tools for human comprehension (2-dimensional syntax, nested-proofs, formal graphics, logicographic symbols, …)*

This is my research menu for the future of THEOREMA (system and company?).

Thanks to Wolfgang Windsteiger!