Using Generative AI in Theoretical CS

AI Transforms Maths Research, University of Augsburg
28 August 2025

Maximilian Doré, University of Oxford
maximilian.dore@cs.ox.ac.uk

What even is Computer Science?

Subject matter

the *mechanisation of thinking*

↑

- planning the steps for cooking a Bolognese
- recognising that a bear is staring into your eyes
- devising a mathematical proof…

Methodology

close to that of mathematics: set-up and use formal language, axiomatise structures that we're interested in, prove properties about these structures, …

→ CS shares methodology with mathematics (ie, stole it from maths); and affects maths by virtue of being about thinking.

The mechanisation of thinking

- Turing: devised mathematical model of what *computers* do

  Turing („computing“) machine: some memory which is manipulated according to a set of rules, called a *program*.
  A problem is *computable* if it can be solved with a program which *stops*.

- Emergence of CS: what’s a good language to program in; what problems are computable; which problems can be solved efficiently; …

- Machine learning: a program which comes up with other programs.

> *Intelligence* is unhelpful notion since it’s supposed to be some innate property.
> *Thinking* is activity—you or a machine can engage in it or not. More workable!
>
> Perhaps helpful analogy: the mechanisation of physical labour

Plan for this talk

- Since research involves some thinking, progress in the mechanisation of thinking affects research.
  - Already happened in maths: calculators, Birch&Swinnerton-Dyer, CAS, …
  - Currently: LLMs and generative AI.
- Explore what this could look like with two case studies of my research:

  §1 Automating higher equalities: Automated reasoning for a new domain

  §2 Devising a logic for resources: More fine-grained programming languages

§1 Automating higher equalities

Equality in type theory

Many current ITP (Agda, Lean, Rocq) are based on dependent type theory.

- Every value has a *type*: $2 : \mathbb{N}$
- Proofs are also just values: $\lambda m, n \to \text{induct}\ (\ldots)\ (\ldots)\ n : \Pi_{m,n:\mathbb{N}}(m + n = n + m)$
- We also want to *stipulate* equalities: $\mathbb{Z}_n$ has values $k : \mathbb{Z}$ such that $\text{eq}_k : k = n + k$ *higher inductive types*
- This gives rise to $\text{eq}_3 : 3 = n + 3$ and $\text{trans}(\text{eq}_3, \text{eq}_{n+3}) : 3 = 2n + 3$, etc.
- If we know that 2 is the multiplicative inverse of 3 in $\mathbb{Z}_5$, then it's also for 8, 13, etc. *coercion*
- Note $\mathbb{Z}_n \cong \text{Fin}\ n$. What if have some proofs about one structure and want to use the other? *univalence*

How can we devise a logic which supports all if this? 💡 Take proof-relevance seriously!
If $p_1, p_2 : x = y$, it's meaningful to study $\alpha, \beta : p_1 = p_2$, $\alpha = \beta$, etc.

Homotopy and Cubical Type Theory

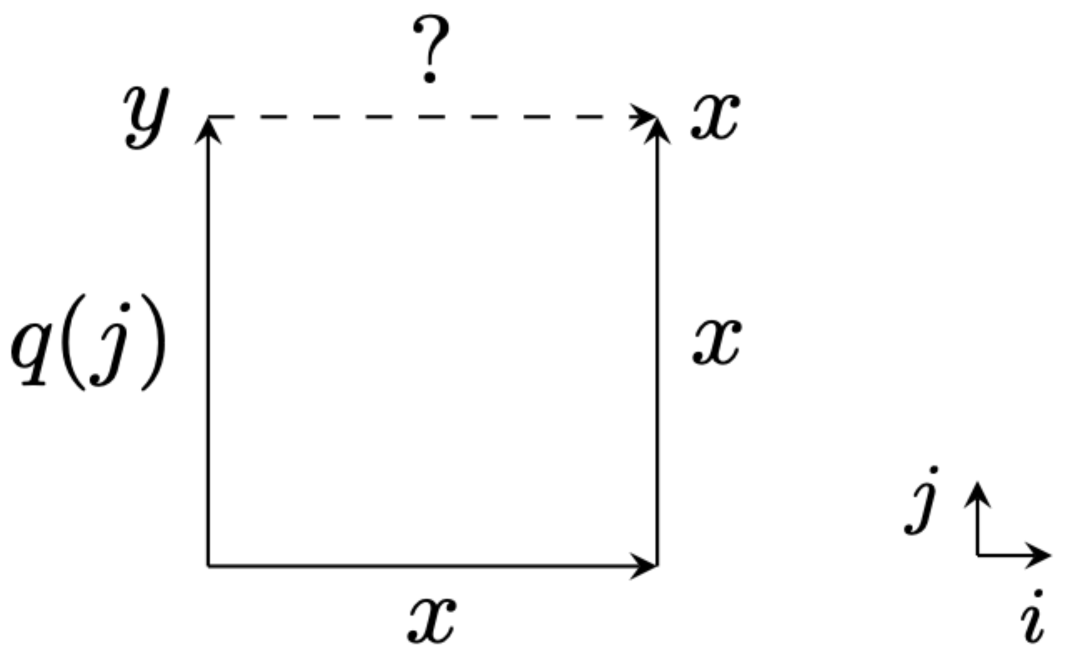
- HoTT: treat *equalities* in type theory akin to *paths* in homotopy theory.
- Cubical Type Theory implements HoTT, taking inspiration from Kan’s cubical sets. Working theorem prover with Cubical Agda.

Equalities are *paths* are *squares/cubes/tesseracts/…*

Suppose $x : A$.

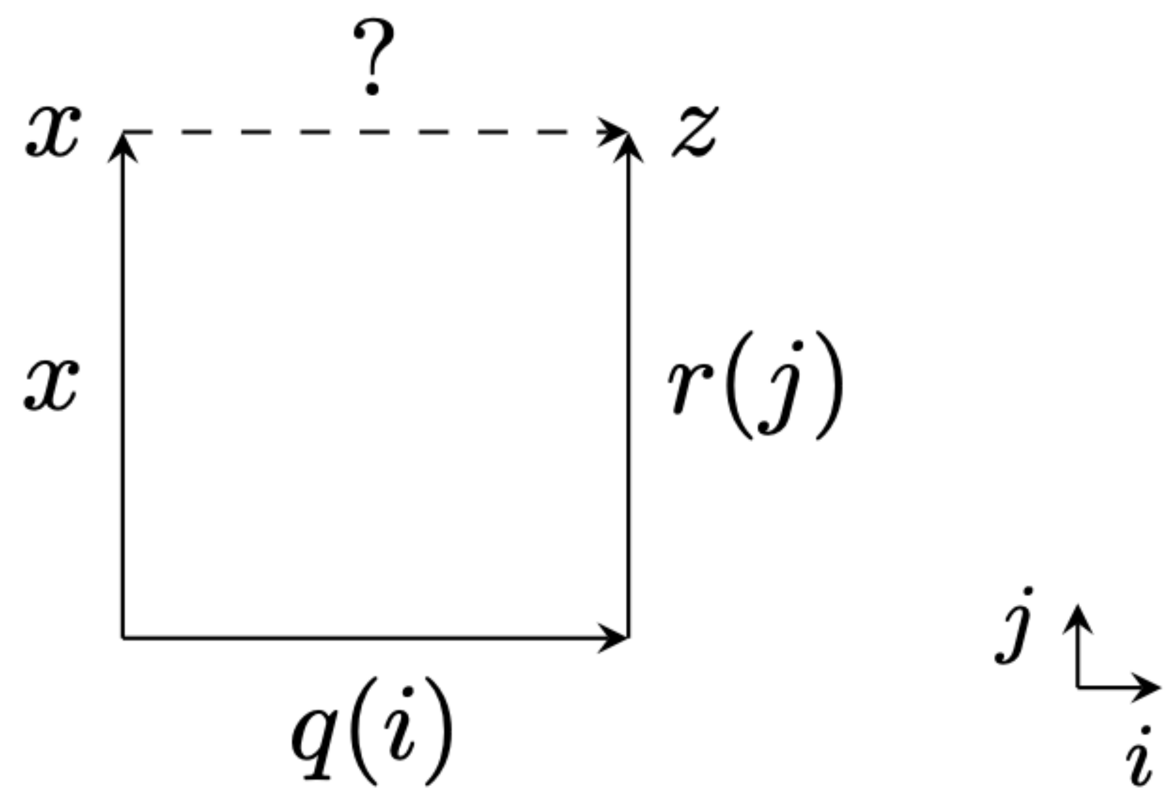
Reflexivity $x = x$ corresponds to the constant path $\lambda i \to x$

Suppose $q : x = y$. For symmetry we have to use *Kan filling:* „any open cube can be filled“



Coming up with equality proofs

Let's show transitivity, ie given $q : x = y$ and $r : y = z$ we want to construct path from x to z.



We can represent a group as a cubical set as follows:

- Base point $\star$
- A loop $a : \star = \star$ for each generator $a \in X$
  - $\lambda i \to \star : \star = \star$ captures the identity element
- A square for each relation

Mechanising Kan fillings

Automating Boundary Filling in Cubical Type Theories
jww Evan Cavallo & Anders Mörtberg
arXiv:2402.12169

- Coming up with Kan fillings is quite tedious, should be automated.
- Have implemented a solver based on *constraint satisfaction programming*.
- Finding cubes which fit together isn't much different from sudoku solving.
- Can find quickly many Kan fillings, also those establishing interesting results like the Eckmann-Hilton argument.

  → old-school mechanisation. Can we use LLMs?

Asking ChatGPT…

Findings from §1

- We have automated some class of proofs. Very *mechanical*.
- Still had to manually invent and implement an algorithm.
- *Generative* AI quite bad at *generating* such proofs.
- Note: research done before (I engaged with) LLMs. Probably would have been useful for implementing the solver.

§2 Devising a logic for resources

Restricting classical logic

- It's often useful to restrict classical logic for some application (eg, in constructive logic any proof is a program)

- **Linear logic** treats variables as resources. $A \otimes B \not\multimap A \qquad A \not\multimap A \otimes A$

  Useful in quantum computing, concurrency, memory management, …

Problem with linear logic when programming: resource usage often not static.

Consider $\mathsf{ifthenelse} : \mathsf{Bool} \to (x : A) \to (y : A) \to A$. How to treat this linearly?

We sometimes use x and sometimes y, depending on the given Bool.

Hacking logics in Agda

- *~Dependent~* type theory allows types to depend on values!
- Initial idea: embed linear logic in type theory (taking inspiration from Gödel's Dialectica construction)
- Works to some extent, but function types missing. Need to stipulate another rule.

  … loads of trial and error…

- Crucial axiom necessary to make logic work discovered by playing around.
- Refactoring, rephrasing and simplifying easy in a proof assistant.

Dependent resources

Dependent Multiplicities in Dependent Linear Type Theory
arXiv:2507.08759

- We obtain a practical programming language with dynamic resource annotations:

$$\mathsf{ifthenelse} : \langle\, b : \mathsf{Bool}\,\rangle \multimap \langle\, A\,\rangle^{\wedge}\,|\,b\,| \multimap \langle\, A\,\rangle^{\wedge}\,|\,\neg\, b\,| \multimap A$$

- Useful for functional programming:

$$\mathsf{map} : \langle\, xs : \mathsf{List}\ A\,\rangle \multimap \langle\, f : \langle\, x : A\,\rangle \multimap B\,\rangle^{\wedge}\ \mathsf{length}\ xs \multimap \mathsf{List}\ B$$

- Agda was crucial to „guide“ thought process.
- Where do LLMs come in?

Asking ChatGPT…

Findings from §2

- Proof assistants like Agda can act as a logical framework in which we can play around with axioms and logics. Strong type discipline weeds out non-sensical things, thereby providing helpful guardrails.
- LLMs helpful when inquiring about well-understood and well-documented research areas
- But also helpful for conceptual work! We can ask about motivation
- (Still: bad at doing things correctly…)
- But can be source of inspiration!

Conclusions

- Instead of artificial intelligence I find it helpful to understand CS and ML as being about the *mechanisation of thinking*.
- Many methods, languages and tools are useful for mechanising research.
  - §1 Generating proofs with hand-written algorithms.
  - §2 Guidance by type-checker when defining something new.
  - §2 LLMs useful when trying to frame and motivate research.
- Surprisingly (to me), LLMs were most helpful for conceptual work and understanding, and not for helping with technical stuff that at first glance seems *mechanical* and apt for computers.