The IMO Grand Challenge

Daniel Selsam

Microsoft Research

September 16th, 2020

Outline

The Great Myth

Third paragraph of standard *Deep Learning* textbook:

The Great Myth

Third paragraph of standard *Deep Learning* textbook:

> *In the early days of artificial intelligence, the field rapidly tackled <u>and solved</u> problems that are intellectually difficult for human beings but relatively straightforward for computers—problems that can be described by a list of formal, mathematical rules. The <u>true challenge</u> to artificial intelligence <u>proved to be</u> solving the tasks that are easy for people to perform but hard for people to describe formally—problems that we solve intuitively, that feel automatic, like recognizing spoken words or faces in images.*

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

The Sad Truth

The Sad Truth

- **Nowhere near** human-level even on formally specified problems.

The Sad Truth

- **Nowhere near** human-level even on formally specified problems.

- Computers are only superhuman in certain niches.
  - problems with simple algorithms (e.g. differentiation)
  - problems with limited structure (e.g. SAT)
  - problems in certain FO theories (e.g. EUF, LRA)
  - (others)

- **Nowhere near** human-level even on formally specified problems.

- Computers are only superhuman in certain niches.
  - problems with simple algorithms (e.g. differentiation)
  - problems with limited structure (e.g. SAT)
  - problems in certain FO theories (e.g. EUF, LRA)
  - (others)

- Such feats have masked the lack of progress on the general problem.

The Sad Truth

- **Nowhere near** human-level even on formally specified problems.

- Computers are only superhuman in certain niches.
  - problems with simple algorithms (e.g. differentiation)
  - problems with limited structure (e.g. SAT)
  - problems in certain FO theories (e.g. EUF, LRA)
  - (others)

- Such feats have masked the lack of progress on the general problem.

- Can still be hard to *produce* machine-checkable proofs at all.
  - even for relatively obvious steps
  - after building libraries of abstractions/tactics
  - with real-time interaction and feedback
  - after decades of tool-building
  - in both mathematics and software verification

Working Forwards

Working Forwards

- The norm in AR is to work forwards from existing methods.
  - better heuristics for existing search spaces
  - more efficient datastructures for existing algorithms
  - procedures for new theories that can slot into SMT solvers

Working Forwards

- The norm in AR is to work forwards from existing methods.
  - better heuristics for existing search spaces
  - more efficient datastructures for existing algorithms
  - procedures for new theories that can slot into SMT solvers

- Obvious pros:
  - can often push technologies much further than expected
  - again and again, cross thresholds that unlock important applications

Working Forwards

- The norm in AR is to work forwards from existing methods.
  - better heuristics for existing search spaces
  - more efficient datastructures for existing algorithms
  - procedures for new theories that can slot into SMT solvers
- Obvious pros:
  - can often push technologies much further than expected
  - again and again, cross thresholds that unlock important applications
- But in the long run: too easy to ignore the ultimate brickwalls!

Working Forwards

- The norm in AR is to work forwards from existing methods.
  - better heuristics for existing search spaces
  - more efficient datastructures for existing algorithms
  - procedures for new theories that can slot into SMT solvers
- Obvious pros:
  - can often push technologies much further than expected
  - again and again, cross thresholds that unlock important applications
- But in the long run: too easy to ignore the ultimate brickwalls!
- **Existing paradigms will never get us to human-level reasoning.**

Working Backwards

Working Backwards

- Working backwards from a goal is **not** a panacea.
  - it may be wholly unclear how to make any progress at all
  - or: it may be impossible to even measure progress
  - or worse: it may just require a lot of one-off engineering

Working Backwards

- Working backwards from a goal is **not** a panacea.
  - it may be wholly unclear how to make any progress at all
  - or: it may be impossible to even measure progress
  - or worse: it may just require a lot of one-off engineering

- But: the **right** goal at the **right** time can be powerful.
  - can trigger the right questions
  - suggest promising new approaches
  - bring together siloed subfields
  - at best: can lead to revolutionary advances!

Working Backwards

- Working backwards from a goal is **not** a panacea.
    - it may be wholly unclear how to make any progress at all
    - or: it may be impossible to even measure progress
    - or worse: it may just require a lot of one-off engineering

- But: the **right** goal at the **right** time can be powerful.
    - can trigger the right questions
    - suggest promising new approaches
    - bring together siloed subfields
    - at best: can lead to revolutionary advances!

- Claim: The IMO is the **right goal** at the **right time**.

Outline

- The most celebrated intellectual competition in the world.

The International Mathematical Olympiad (IMO)

- The most celebrated intellectual competition in the world.

- Logistics:
  - every year, $>$100 countries train and filter and send 6 (HS) students.
  - two-day test, with 4.5 hours/3 problems each day
  - medals are percentile-based: top $\approx$8% win Gold

The International Mathematical Olympiad (IMO)

- The most celebrated intellectual competition in the world.

- Logistics:
  - every year, $>$100 countries train and filter and send 6 (HS) students.
  - two-day test, with 4.5 hours/3 problems each day
  - medals are percentile-based: top $\approx$8% win Gold

- All material is elementary.
  - only high-school level mathematics required
  - algebra, number theory, combinatorics, geometry
  - solutions tend to be short and sweet

The International Mathematical Olympiad (IMO)

- The most celebrated intellectual competition in the world.

- Logistics:
  - every year, $>$100 countries train and filter and send 6 (HS) students.
  - two-day test, with 4.5 hours/3 problems each day
  - medals are percentile-based: top $\approx$8% win Gold

- All material is elementary.
  - only high-school level mathematics required
  - algebra, number theory, combinatorics, geometry
  - solutions tend to be short and sweet
  - **but: they are designed to require tremendous ingenuity**

The International Mathematical Olympiad (IMO)

- The most celebrated intellectual competition in the world.

- Logistics:
  - every year, $>$100 countries train and filter and send 6 (HS) students.
  - two-day test, with 4.5 hours/3 problems each day
  - medals are percentile-based: top $\approx$8% win Gold

- All material is elementary.
  - only high-school level mathematics required
  - algebra, number theory, combinatorics, geometry
  - solutions tend to be short and sweet
  - **but: they are designed to require tremendous ingenuity**

- Extremely elite.

Problem (IMO 2005 #3)

Let x, y, z be three positive reals such that $xyz \geq 1$. Prove that

$$\frac{x^5 - x^2}{x^5 + y^2 + z^2} + \frac{y^5 - y^2}{x^2 + y^5 + z^2} + \frac{z^5 - z^2}{x^2 + y^2 + z^5} \geq 0$$

Example Problems (Number Theory)

Problem (IMO 2003 #6)

Show that for each prime p, there exists a prime q such that $n^p - p$ is not divisible by q for any positive integer n.

Problem (IMO 1995 #6)

Let p be an odd prime number. How many p-element subsets A of $\{1, 2, \ldots, 2p\}$ are there, the sum of whose elements is divisible by p?

Problem (IMO 2006 #6)

Assign to each side b of a convex polygon P the maximum area of a triangle that has b as a side and is contained in P. Show that the sum of the areas assigned to the sides of P is at least twice the area of P.

- **The challenge:**

- **The challenge: build an AI that can win a gold medal.**

The IMO Grand Challenge

- **The challenge: build an AI that can win a gold medal.**
- Formal-to-formal (F2F) variant of the IMO.
  - AI receives formal statements of problems
  - must produce machine-checkable proofs
  - (caveat: “determine” problems)

The IMO Grand Challenge

- **The challenge: build an AI that can win a gold medal.**

- Formal-to-formal (F2F) variant of the IMO.
  - AI receives formal statements of problems
  - must produce machine-checkable proofs
  - (caveat: "determine" problems)

- Other details:
  - system must be checksummed before the problems are released
  - no access to Internet
  - regular wall-clock time but no other computational limitations
  - proofs must be checkable in (say) 10 minutes
  - (roughly what it takes to check a human proof)

The IMO Grand Challenge

- **The challenge: build an AI that can win a gold medal.**
- Formal-to-formal (F2F) variant of the IMO.
  - AI receives formal statements of problems
  - must produce machine-checkable proofs
  - (caveat: “determine” problems)
- Other details:
  - system must be checksummed before the problems are released
  - no access to Internet
  - regular wall-clock time but no other computational limitations
  - proofs must be checkable in (say) 10 minutes
  - (roughly what it takes to check a human proof)
- Committee:
  - Leonardo de Moura (MSR)
  - Kevin Buzzard (Imperial College London)
  - Reid Barton (University of Pittsburgh)
  - Percy Liang (Stanford University)
  - Sarah Loos (Apple)
  - Freek Wiedijk (University of Nijmegen)

Why the IMO?

Why the IMO?

- Extremely simple setting:
  - problems are formally specified (no vagueness or ambiguity)
  - solutions can be machine-checked (no need to imitate humans)
  - closed-world (limited background knowledge required)

Why the IMO?

- Extremely simple setting:
  - problems are formally specified (no vagueness or ambiguity)
  - solutions can be machine-checked (no need to imitate humans)
  - closed-world (limited background knowledge required)
- Yet broad consensus:
  - incredibly hard, maybe even AI-complete
  - would be among all-time great achievements of CS
  - winning tech would revolutionize AI, AR, PL, mathematics

Why the IMO?

- Extremely simple setting:
  - problems are formally specified (no vagueness or ambiguity)
  - solutions can be machine-checked (no need to imitate humans)
  - closed-world (limited background knowledge required)
- Yet broad consensus:
  - incredibly hard, maybe even AI-complete
  - would be among all-time great achievements of CS
  - winning tech would revolutionize AI, AR, PL, mathematics
- Ongoing supply of new problems.
  - long-standing, global, decentralized process

Why the IMO?

- Extremely simple setting:
  - problems are formally specified (no vagueness or ambiguity)
  - solutions can be machine-checked (no need to imitate humans)
  - closed-world (limited background knowledge required)
- Yet broad consensus:
  - incredibly hard, maybe even AI-complete
  - would be among all-time great achievements of CS
  - winning tech would revolutionize AI, AR, PL, mathematics
- Ongoing supply of new problems.
  - long-standing, global, decentralized process
- Well-defined notion of success: winning a gold medal.

Why the IMO?

- Extremely simple setting:
  - problems are formally specified (no vagueness or ambiguity)
  - solutions can be machine-checked (no need to imitate humans)
  - closed-world (limited background knowledge required)
- Yet broad consensus:
  - incredibly hard, maybe even AI-complete
  - would be among all-time great achievements of CS
  - winning tech would revolutionize AI, AR, PL, mathematics
- Ongoing supply of new problems.
  - long-standing, global, decentralized process
- Well-defined notion of success: winning a gold medal.
- **Most importantly:**

Why the IMO?

- Extremely simple setting:
  - problems are formally specified (no vagueness or ambiguity)
  - solutions can be machine-checked (no need to imitate humans)
  - closed-world (limited background knowledge required)
- Yet broad consensus:
  - incredibly hard, maybe even AI-complete
  - would be among all-time great achievements of CS
  - winning tech would revolutionize AI, AR, PL, mathematics
- Ongoing supply of new problems.
  - long-standing, global, decentralized process
- Well-defined notion of success: winning a gold medal.
- **Most importantly: we think we have a real chance!**

Why the IMO?

- Extremely simple setting:
  - problems are formally specified (no vagueness or ambiguity)
  - solutions can be machine-checked (no need to imitate humans)
  - closed-world (limited background knowledge required)
- Yet broad consensus:
  - incredibly hard, maybe even AI-complete
  - would be among all-time great achievements of CS
  - winning tech would revolutionize AI, AR, PL, mathematics
- Ongoing supply of new problems.
  - long-standing, global, decentralized process
- Well-defined notion of success: winning a gold medal.
- **Most importantly: we think we have a real chance!**
  - but we need to work together as community

Why the IMO?

- Extremely simple setting:
  - problems are formally specified (no vagueness or ambiguity)
  - solutions can be machine-checked (no need to imitate humans)
  - closed-world (limited background knowledge required)
- Yet broad consensus:
  - incredibly hard, maybe even AI-complete
  - would be among all-time great achievements of CS
  - winning tech would revolutionize AI, AR, PL, mathematics
- Ongoing supply of new problems.
  - long-standing, global, decentralized process
- Well-defined notion of success: winning a gold medal.
- **Most importantly: we think we have a real chance!**
  - but we need to work together as community
  - and we need to play the long game

Outline

High-Level Strategy

High-Level Strategy

1. Formalize historical problems in Lean.
   - grassroots effort in Mathlib community even before IMO-GC
   - many former winners are involved
   - most of the background math is already there

High-Level Strategy

1. Formalize historical problems in Lean.
   - grassroots effort in Mathlib community even before IMO-GC
   - many former winners are involved
   - most of the background math is already there

2. Compress proofs using **very high level** tactics.
   - the kinds of strategies that humans are taught
   - e.g. small-n, symmetry, extremes, invariants, pigeonhole
   - **challenge**: how to manifest these in software?

High-Level Strategy

1. Formalize historical problems in Lean.
   - grassroots effort in Mathlib community even before IMO-GC
   - many former winners are involved
   - most of the background math is already there

2. Compress proofs using **very high level** tactics.
   - the kinds of strategies that humans are taught
   - e.g. small-n, symmetry, extremes, invariants, pigeonhole
   - **challenge**: how to manifest these in software?

3. Train neural networks to guide search.
   - VHL tactics will be riddled with choice points
   - no way to hand-engineer all the low-level heuristics
   - **challenge**: how to learn heuristics from few examples?

High-Level Strategy

1. Formalize historical problems in Lean.
   - grassroots effort in Mathlib community even before IMO-GC
   - many former winners are involved
   - most of the background math is already there

2. Compress proofs using **very high level** tactics.
   - the kinds of strategies that humans are taught
   - e.g. small-n, symmetry, extremes, invariants, pigeonhole
   - **challenge**: how to manifest these in software?

3. Train neural networks to guide search.
   - VHL tactics will be riddled with choice points
   - no way to hand-engineer all the low-level heuristics
   - **challenge**: how to learn heuristics from few examples?

4. Finish the job with armada of search.

Outline

Outline

- Standard advice for talks: stick to the past.

Outline

- Standard advice for talks: stick to the past.
- Contra advice: rest of talk is preliminary roadmap.

Outline

- Standard advice for talks: stick to the past.
- Contra advice: rest of talk is preliminary roadmap.
  - potential solutions to the two main challenges

Outline

- Standard advice for talks: stick to the past.

- Contra advice: rest of talk is preliminary roadmap.
  - potential solutions to the two main challenges
  - warning: ideas reasonably fleshed out but far from battle-tested

Outline

- Standard advice for talks: stick to the past.

- Contra advice: rest of talk is preliminary roadmap.
  - potential solutions to the two main challenges
  - warning: ideas reasonably fleshed out but far from battle-tested

- Two interrelated WIP ideas:
  - representing strategies with the **search transformer**
  - guiding search with the **universal oracle**

Outline

- Standard advice for talks: stick to the past.
- Contra advice: rest of talk is preliminary roadmap.
  - potential solutions to the two main challenges
  - warning: ideas reasonably fleshed out but far from battle-tested
- Two interrelated WIP ideas:
  - representing strategies with the **search transformer**
  - guiding search with the **universal oracle**
- The real War Machine that makes these projects possible:

Outline

- Standard advice for talks: stick to the past.

- Contra advice: rest of talk is preliminary roadmap.
  - potential solutions to the two main challenges
  - warning: ideas reasonably fleshed out but far from battle-tested

- Two interrelated WIP ideas:
  - representing strategies with the **search transformer**
  - guiding search with the **universal oracle**

- The real War Machine that makes these projects possible: **Lean4**.

Outline

- Standard advice for talks: stick to the past.
- Contra advice: rest of talk is preliminary roadmap.
  - potential solutions to the two main challenges
  - warning: ideas reasonably fleshed out but far from battle-tested
- Two interrelated WIP ideas:
  - representing strategies with the **search transformer**
  - guiding search with the **universal oracle**
- The real War Machine that makes these projects possible: **Lean4**.
  - similar logic as battle-tested by Mathlib

Outline

- Standard advice for talks: stick to the past.

- Contra advice: rest of talk is preliminary roadmap.
  - potential solutions to the two main challenges
  - warning: ideas reasonably fleshed out but far from battle-tested

- Two interrelated WIP ideas:
  - representing strategies with the **search transformer**
  - guiding search with the **universal oracle**

- The real War Machine that makes these projects possible: **Lean4**.
  - similar logic as battle-tested by Mathlib
  - new in Lean4: real programming language, ridiculous performance

Outline

- Standard advice for talks: stick to the past.

- Contra advice: rest of talk is preliminary roadmap.
  - potential solutions to the two main challenges
  - warning: ideas reasonably fleshed out but far from battle-tested

- Two interrelated WIP ideas:
  - representing strategies with the **search transformer**
  - guiding search with the **universal oracle**

- The real War Machine that makes these projects possible: **Lean4**.
  - similar logic as battle-tested by Mathlib
  - new in Lean4: real programming language, ridiculous performance
  - (no need to drop down to C++ for perf-critical tactics)

Outline

- Standard advice for talks: stick to the past.
- Contra advice: rest of talk is preliminary roadmap.
  - potential solutions to the two main challenges
  - warning: ideas reasonably fleshed out but far from battle-tested
- Two interrelated WIP ideas:
  - representing strategies with the **search transformer**
  - guiding search with the **universal oracle**
- The real War Machine that makes these projects possible: **Lean4**.
  - similar logic as battle-tested by Mathlib
  - new in Lean4: real programming language, ridiculous performance
  - (no need to drop down to C++ for perf-critical tactics)
  - built by Leonardo de Moura (MSR) and Sebastian Ullrich (KIT)

Outline

Tactics, Not Agents

Tactics, Not Agents

- Standard agent/environment model for ITP:
  - $(\text{Theorems}, \text{Goal}, \text{Action}) \to [\text{Goal}]$
  - loop:
    - look at theorems, current goal, possible actions
    - select action, apply it
    - add resulting subgoals to goal stack

Tactics, Not Agents

- Standard agent/environment model for ITP:
  - $(\text{Theorems}, \text{Goal}, \text{Action}) \to [\text{Goal}]$
  - loop:
    - look at theorems, current goal, possible actions
    - select action, apply it
    - add resulting subgoals to goal stack
- Appealing, but has limitations.
  - binary distinction between choices and black-box tactics
  - in much of formal math, the line is very blurred

Tactics, Not Agents

- Standard agent/environment model for ITP:
  - $(\text{Theorems}, \text{Goal}, \text{Action}) \to [\text{Goal}]$
  - loop:
    - look at theorems, current goal, possible actions
    - select action, apply it
    - add resulting subgoals to goal stack
- Appealing, but has limitations.
  - binary distinction between choices and black-box tactics
  - in much of formal math, the line is very blurred
- **Tactics are computer programs, not atomic actions.**
  - keep their own kind of state (not necessarily just list of goals)
  - may make **internal** heuristic decisions
  - may call other tactics recursively
  - compositionality is where their power comes from!

Tactics, Not Agents

- Standard agent/environment model for ITP:
  - $(\mathrm{Theorems}, \mathrm{Goal}, \mathrm{Action}) \to [\mathrm{Goal}]$
  - loop:
    - look at theorems, current goal, possible actions
    - select action, apply it
    - add resulting subgoals to goal stack
- Appealing, but has limitations.
  - binary distinction between choices and black-box tactics
  - in much of formal math, the line is very blurred
- **Tactics are computer programs, not atomic actions.**
  - keep their own kind of state (not necessarily just list of goals)
  - may make **internal** heuristic decisions
  - may call other tactics recursively
  - compositionality is where their power comes from!

Roadmap I: New agent/environment model

Write nondeterministic tactics with explicit choice points; agent's job is to execute these tactics, choosing which branches to go down at each choice point.

Nondeterministic Tactics

Nondeterministic Tactics

- Status quo: regular tactics **hardcode** choice-point ordering.
  - `f <|> g` means "try f, if it fails, try g"
  - search space and search decisions intertwined

Nondeterministic Tactics

- Status quo: regular tactics **hardcode** choice-point ordering.
  - `f <|> g` means "try f, if it fails, try g"
  - search space and search decisions intertwined
- Our approach: **reify** the choice points.
  - factor out heuristics from search space
  - allow multiple, modular ways of *guiding* tactics

Nondeterministic Tactics

- Status quo: regular tactics **hardcode** choice-point ordering.
  - `f <|> g` means "try *f*, if it fails, try *g*"
  - search space and search decisions intertwined
- Our approach: **reify** the choice points.
  - factor out heuristics from search space
  - allow multiple, modular ways of *guiding* tactics
- Silly example (more details to come):

```
blindRewrite : NondeterministicTactic := do
  h <- choose env.theorems
  execute (rewrite h)
```

Nondeterministic Tactics

- Status quo: regular tactics **hardcode** choice-point ordering.
  - `f <|> g` means "try *f*, if it fails, try *g*"
  - search space and search decisions intertwined
- Our approach: **reify** the choice points.
  - factor out heuristics from search space
  - allow multiple, modular ways of *guiding* tactics
- Silly example (more details to come):

```
blindRewrite : NondeterministicTactic := do
  h <- choose env.theorems
  execute (rewrite h)

breadthFirstSearch blindRewrite
depthFirstSearch blindRewrite
```

Nondeterministic Tactics

- Status quo: regular tactics **hardcode** choice-point ordering.
  - `f <|> g` means "try *f*, if it fails, try *g*"
  - search space and search decisions intertwined
- Our approach: **reify** the choice points.
  - factor out heuristics from search space
  - allow multiple, modular ways of *guiding* tactics
- Silly example (more details to come):

```
blindRewrite : NondeterministicTactic := do
  h <- choose env.theorems
  execute (rewrite h)

breadthFirstSearch blindRewrite
depthFirstSearch blindRewrite
```

- Open question: how best to encode IMO strategies?
  - extreme 1: detailed proof scripts (no search)
  - extreme 2: choose bits of proof (insane search)
  - **obviously: we want something in the middle**

Example: Olympiad Inequalities

Problem (JBMO 2002)

Let $a, b, c > 0$ *and prove that:*

$$2(\sum_{cyc} a)^2(\sum_{cyc} \frac{1}{a(a+b)}) \geq 27$$

Problem (JBMO 2002)

Let $a, b, c > 0$ *and prove that:*

$$2(\sum_{cyc} a)^2(\sum_{cyc} \frac{1}{a(a+b)}) \geq 27$$

Calculational proof:

$$\begin{aligned}
& 2(\sum_{cyc} a)^2(\sum_{cyc} \frac{1}{a(a+b)}) \\
&= \left(\sum_{cyc} a\right)\left(\sum_{cyc} 2a\right)\left(\sum_{cyc} \frac{1}{a(a+b)}\right) && \text{(group)} \\
&= \left(\sum_{cyc} a\right)\left(\sum_{cyc} a+b\right)\left(\sum_{cyc} \frac{1}{a(a+b)}\right) && \text{(cycle)} \\
&\geq \left(\sum_{cyc} \left(\frac{a(a+b)}{a(a+b)}\right)\right)^3 && \text{(Holder)} \\
&= \left(\sum_{cyc} 1\right)^3 && \text{(cancel)} \\
&= 27 && \text{(eval)}
\end{aligned}$$

Example: Olympiad Inequalities

Problem (JBMO 2002)

Let $a, b, c > 0$ and prove that:

$$2(\sum_{cyc} a)^2(\sum_{cyc} \frac{1}{a(a+b)}) \geq 27$$

Calculational proof:

$$\begin{aligned}
&2(\sum_{cyc} a)^2(\sum_{cyc} \frac{1}{a(a+b)}) \\
&= \left(\sum_{cyc} a\right)\left(\sum_{cyc} 2a\right)\left(\sum_{cyc} \frac{1}{a(a+b)}\right) && \text{(group)} \\
&= \left(\sum_{cyc} a\right)\left(\sum_{cyc} a+b\right)\left(\sum_{cyc} \frac{1}{a(a+b)}\right) && \text{(cycle)} \\
&\geq \left(\sum_{cyc} \left(\frac{a(a+b)}{a(a+b)}\right)\right)^3 && \text{(Holder)} \\
&= \left(\sum_{cyc} 1\right)^3 && \text{(cancel)} \\
&= 27 && \text{(eval)}
\end{aligned}$$

High-level proof: make LHS look like LHS of Holder's, then apply it.

- Easy to implement nondeterministic strategy that can prove it:

Example: Olympiad Inequalities

- Easy to implement nondeterministic strategy that can prove it:

```
abstractProveJBMO2002 := do
  thm <- choose standardDozen
  makeLookLike (getLHS goal) (getLHS thm)
  apply thm
  finish
```

Example: Olympiad Inequalities

- Easy to implement nondeterministic strategy that can prove it:

```
abstractProveJBMO2002 := do
  thm <- choose standardDozen
  makeLookLike (getLHS goal) (getLHS thm)
  apply thm
  finish
```

- May be hard to specify:
  - which theorem to try next?
  - how to `makeLookLike` one term into another?

Example: Olympiad Inequalities

- Easy to implement nondeterministic strategy that can prove it:

```
abstractProveJBMO2002 := do
  thm <- choose standardDozen
  makeLookLike (getLHS goal) (getLHS thm)
  apply thm
  finish
```

- May be hard to specify:
  - which theorem to try next?
  - how to `makeLookLike` one term into another?
- But, simple script already extremely useful!
  - `makeLookLike` gets a **specification/goal**
  - can use target to prune search space dramatically

Example: Olympiad Inequalities

- Easy to implement nondeterministic strategy that can prove it:

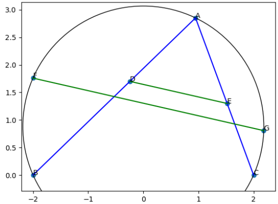
```
abstractProveJBMO2002 := do
  thm <- choose standardDozen
  makeLookLike (getLHS goal) (getLHS thm)
  apply thm
  finish
```

- May be hard to specify:
  - which theorem to try next?
  - how to `makeLookLike` one term into another?
- But, simple script already extremely useful!
  - `makeLookLike` gets a **specification/goal**
  - can use target to prune search space dramatically
- Easy to relax proof further:
  - `getLHS goal` → `choose (subterms goal)`
  - `apply` → `rewrite`
  - `finish` → `simplify, recurse`

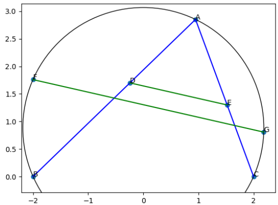
Example: Geometry

Example: Geometry

IMO 2018 Problem 1:

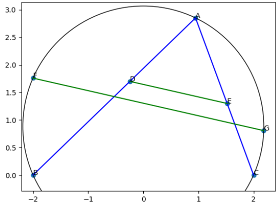


IMO 2018 Problem 1:



- Most Geometry proofs require introducing *auxiliary constructions*.
  - *e.g.* midpoints, feet, intersections, reflections, completions, etc.
  - large (indeed, infinite) set of possibilities

IMO 2018 Problem 1:

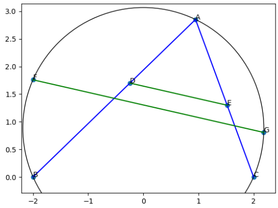


- Most Geometry proofs require introducing *auxiliary constructions*.
  - *e.g.* midpoints, feet, intersections, reflections, completions, etc.
  - large (indeed, infinite) set of possibilities

- (Start of human proof) Let M and N be the arc-midpoints of AB and AC respectively. It suffices to show that $\overline{FG} \| \overline{MN}$ and $\overline{DE} \| \overline{MN}$.

Example: Geometry

IMO 2018 Problem 1:



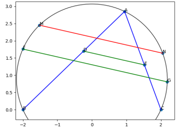
- Most Geometry proofs require introducing *auxiliary constructions*.
  - *e.g.* midpoints, feet, intersections, reflections, completions, etc.
  - large (indeed, infinite) set of possibilities
- (Start of human proof) Let M and N be the arc-midpoints of AB and AC respectively. It suffices to show that $\overline{FG} \| \overline{MN}$ and $\overline{DE} \| \overline{MN}$.
- Ho, what magic?
  - how do you know to try M and N?
  - what is the abstract strategy?

Example: Geometry

- Answer: look at the diagram!

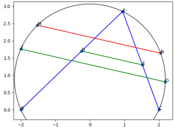
Example: Geometry

- Answer: look at the diagram!



Example: Geometry

- Answer: look at the diagram!

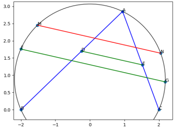


- Simple nondeterministic strategy:

```
abstractProveGeo := do
  thm <- choose geoTheorems
  apply thm
  when (hasVariables goal) (do points <- chooseFromModel; instantiate points)
  abstractProveGeo
```

Example: Geometry

- Answer: look at the diagram!



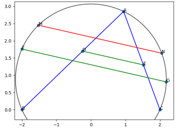
- Simple nondeterministic strategy:

```
abstractProveGeo := do
  thm <- choose geoTheorems
  apply thm
  when (hasVariables goal) (do points <- chooseFromModel; instantiate points)
  abstractProveGeo
```

- No idea how to specify:
  - which theorem to try next?
  - which of the promising constructions to try next?

Example: Geometry

- Answer: look at the diagram!



- Simple nondeterministic strategy:

```
abstractProveGeo := do
  thm <- choose geoTheorems
  apply thm
  when (hasVariables goal) (do points <- chooseFromModel; instantiate points)
  abstractProveGeo
```

- No idea how to specify:
  - which theorem to try next?
  - which of the promising constructions to try next?
- But simple script is extremely useful!
  - candidate constructions pruned by several OOM
  - no loss of power (as long as model is correct)

- **The best tactics will still induce intractable search spaces.**
  - we can only introspect so much
  - we can only provide so much structure before we dull the system

Decisions, Decisions

- **The best tactics will still induce intractable search spaces.**
  - we can only introspect so much
  - we can only provide so much structure before we dull the system

- Can we leverage learning to navigate these spaces?

- **The best tactics will still induce intractable search spaces.**
  - we can only introspect so much
  - we can only provide so much structure before we dull the system

- Can we leverage learning to navigate these spaces?

- Hypothesis: deep learning has failed to advance AR because:
  - search spaces too low-level
  - wrong agent models
  - **and obviously: not enough data**

Decisions, Decisions

- **The best tactics will still induce intractable search spaces.**
  - we can only introspect so much
  - we can only provide so much structure before we dull the system

- Can we leverage learning to navigate these spaces?

- Hypothesis: deep learning has failed to advance AR because:
  - search spaces too low-level
  - wrong agent models
  - **and obviously: not enough data**

Roadmap II: Extreme Genericity

Embed search problems **generically** so that a single neural network can pool data across all conceivable search problems and provide zero-shot guidance.

Pooling Data

Pooling Data

- Want to **pool** training data across many domains:

Pooling Data

- Want to **pool** training data across many domains:
  - IMO problems

Pooling Data

- Want to **pool** training data across many domains:
    - IMO problems
    - Mathlib proper

Pooling Data

- Want to **pool** training data across many domains:
  - IMO problems
  - Mathlib proper
  - other formal math libraries (*e.g.* Metamath)

Pooling Data

- Want to **pool** training data across many domains:
  - IMO problems
  - Mathlib proper
  - other formal math libraries (*e.g.* Metamath)
  - computer algebra (*e.g.* integrals, sums)

Pooling Data

- Want to **pool** training data across many domains:
  - IMO problems
  - Mathlib proper
  - other formal math libraries (*e.g.* Metamath)
  - computer algebra (*e.g.* integrals, sums)
  - synthesis problems (*e.g.* ARC)

Pooling Data

- Want to **pool** training data across many domains:
  - IMO problems
  - Mathlib proper
  - other formal math libraries (*e.g.* Metamath)
  - computer algebra (*e.g.* integrals, sums)
  - synthesis problems (*e.g.* ARC)
  - puzzles (*e.g.* Sudoku)

Pooling Data

- Want to **pool** training data across many domains:
  - IMO problems
  - Mathlib proper
  - other formal math libraries (*e.g.* Metamath)
  - computer algebra (*e.g.* integrals, sums)
  - synthesis problems (*e.g.* ARC)
  - puzzles (*e.g.* Sudoku)
  - verification problems?

Pooling Data

- Want to **pool** training data across many domains:
  - IMO problems
  - Mathlib proper
  - other formal math libraries (*e.g.* Metamath)
  - computer algebra (*e.g.* integrals, sums)
  - synthesis problems (*e.g.* ARC)
  - puzzles (*e.g.* Sudoku)
  - verification problems?
  - code optimizers?

Pooling Data

- Want to **pool** training data across many domains:
  - IMO problems
  - Mathlib proper
  - other formal math libraries (*e.g.* Metamath)
  - computer algebra (*e.g.* integrals, sums)
  - synthesis problems (*e.g.* ARC)
  - puzzles (*e.g.* Sudoku)
  - verification problems?
  - code optimizers?
  - query planners?

Pooling Data

- Want to **pool** training data across many domains:
  - IMO problems
  - Mathlib proper
  - other formal math libraries (*e.g.* Metamath)
  - computer algebra (*e.g.* integrals, sums)
  - synthesis problems (*e.g.* ARC)
  - puzzles (*e.g.* Sudoku)
  - verification problems?
  - code optimizers?
  - query planners?
  - board games?

Pooling Data

- Want to **pool** training data across many domains:
  - IMO problems
  - Mathlib proper
  - other formal math libraries (*e.g.* Metamath)
  - computer algebra (*e.g.* integrals, sums)
  - synthesis problems (*e.g.* ARC)
  - puzzles (*e.g.* Sudoku)
  - verification problems?
  - code optimizers?
  - query planners?
  - board games?
  - ...

Pooling Data

- Want to **pool** training data across many domains:
    - IMO problems
    - Mathlib proper
    - other formal math libraries (*e.g.* Metamath)
    - computer algebra (*e.g.* integrals, sums)
    - synthesis problems (*e.g.* ARC)
    - puzzles (*e.g.* Sudoku)
    - verification problems?
    - code optimizers?
    - query planners?
    - board games?
    - ...
    - (endless possibilities)

Pooling Data

- Want to **pool** training data across many domains:
  - IMO problems
  - Mathlib proper
  - other formal math libraries (*e.g.* Metamath)
  - computer algebra (*e.g.* integrals, sums)
  - synthesis problems (*e.g.* ARC)
  - puzzles (*e.g.* Sudoku)
  - verification problems?
  - code optimizers?
  - query planners?
  - board games?
  - ...
  - (endless possibilities)
  - (empirical/economic question where to draw the line)

Pooling Data

- Want to **pool** training data across many domains:
  - IMO problems
  - Mathlib proper
  - other formal math libraries (*e.g.* Metamath)
  - computer algebra (*e.g.* integrals, sums)
  - synthesis problems (*e.g.* ARC)
  - puzzles (*e.g.* Sudoku)
  - verification problems?
  - code optimizers?
  - query planners?
  - board games?
  - ...
  - (endless possibilities)
  - (empirical/economic question where to draw the line)

- To pool: search problems must be made **commensurable**.

Generic Search Problems

Generic Search Problems

- New abstraction `SearchT` for representing arbitrary search problems.

Generic Search Problems

- New abstraction `SearchT` for representing arbitrary search problems.
- Basic idea: a search problem is an arbitrary program that either:

Generic Search Problems

- New abstraction `SearchT` for representing arbitrary search problems.
- Basic idea: a search problem is an arbitrary program that either:
  - fails

Generic Search Problems

- New abstraction `SearchT` for representing arbitrary search problems.
- Basic idea: a search problem is an arbitrary program that either:
  - fails
  - successfully returns a value

Generic Search Problems

- New abstraction `SearchT` for representing arbitrary search problems.
- Basic idea: a search problem is an arbitrary program that either:
  - fails
  - successfully returns a value
  - returns "choice point"

Generic Search Problems

- New abstraction `SearchT` for representing arbitrary search problems.
- Basic idea: a search problem is an arbitrary program that either:
  - fails
  - successfully returns a value
  - returns "choice point"
- Choice point:
  - user-specified data deemed relevant for decision
  - list of possible choices

Generic Search Problems

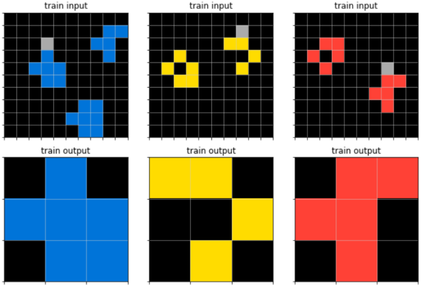
- New abstraction `SearchT` for representing arbitrary search problems.
- Basic idea: a search problem is an arbitrary program that either:
  - fails
  - successfully returns a value
  - returns "choice point"
- Choice point:
  - user-specified data deemed relevant for decision
  - list of possible choices
- Choice:
  - some data summarizing the choice
  - another arbitrary search problem, *i.e.* a continuation

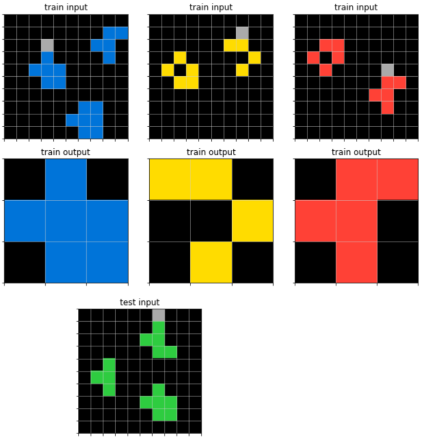
Generic Search Problems

- New abstraction `SearchT` for representing arbitrary search problems.
- Basic idea: a search problem is an arbitrary program that either:
  - fails
  - successfully returns a value
  - returns "choice point"
- Choice point:
  - user-specified data deemed relevant for decision
  - list of possible choices
- Choice:
  - some data summarizing the choice
  - another arbitrary search problem, *i.e.* a continuation
- Can "run" a `SearchT` program in variety of generic ways.
  - depth-first search
  - breadth-first search
  - later: heuristic search

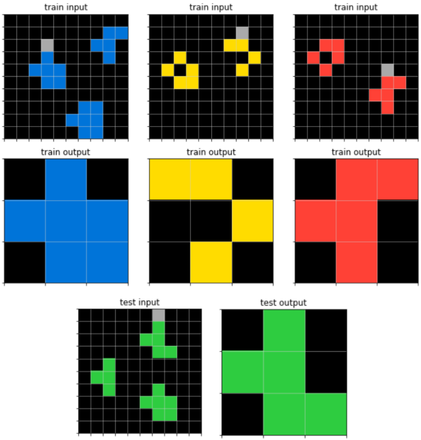
Example: ARC

Example: ARC


train input
train input
train input
train output
train output
train output

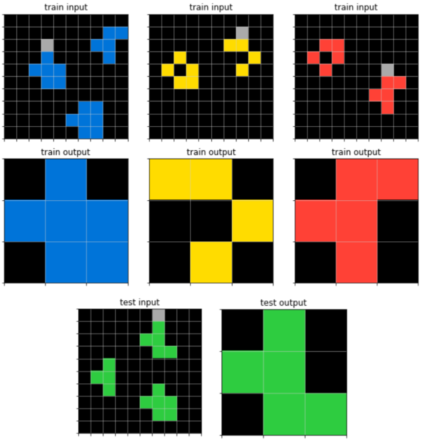

train input
train input
train input
train output
train output
train output
test input

Example: ARC



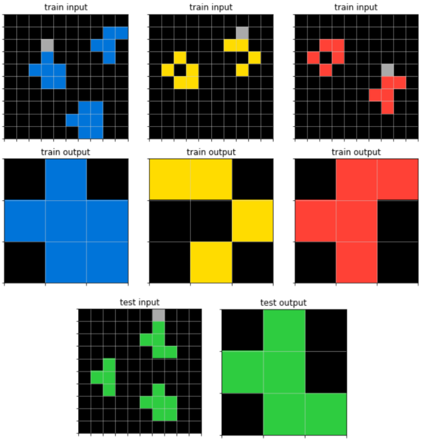
train input
train input
train input
train output
train output
train output
test input
test output

Example: ARC



High-level solution:

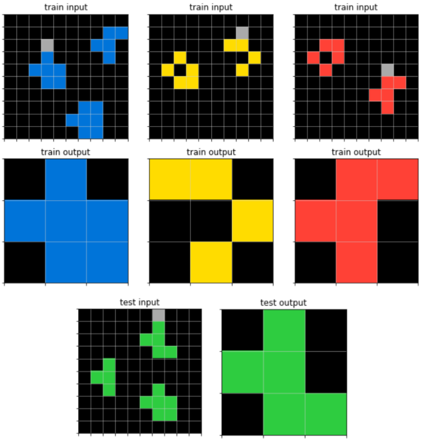
Example: ARC



High-level solution:

- **split input** into shapes by color and connectivity

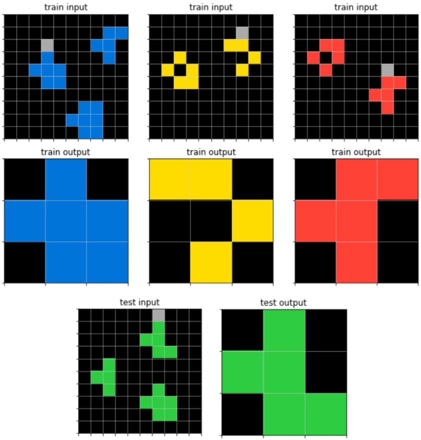
Example: ARC



High-level solution:

- **split input** into shapes by color and connectivity
- **find the special** shape that touches a grey cell

Example: ARC

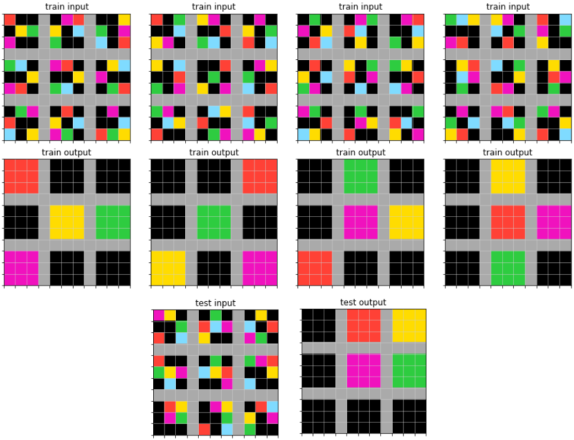


High-level solution:

- **split input** into shapes by color and connectivity
- **find the special** shape that touches a grey cell
- **guess** the smallest square containing the special shape

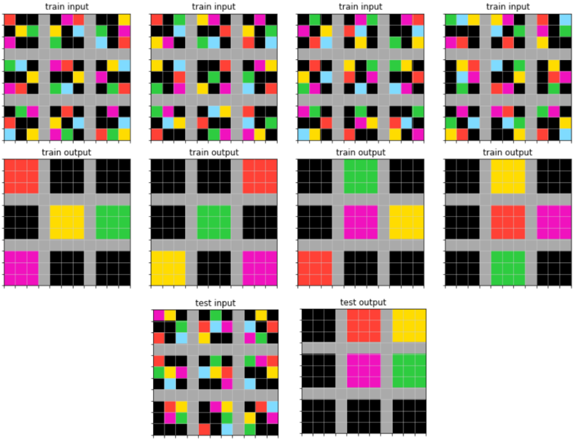
Example: ARC

Example: ARC



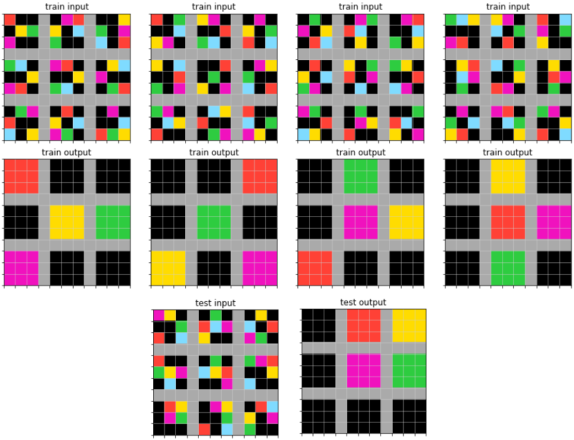
train input
train input
train input
train input
train output
train output
train output
train output
test input
test output

Example: ARC



High-level solution:

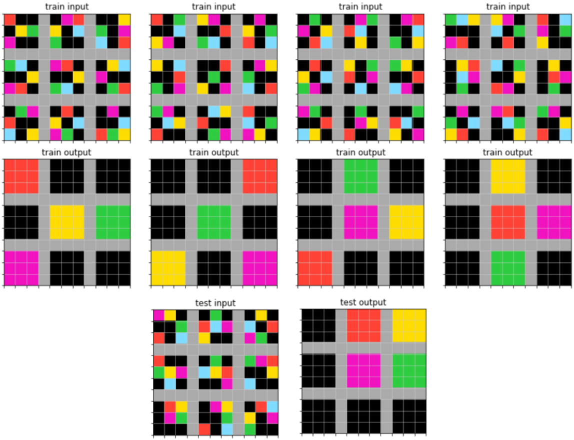
Example: ARC



High-level solution:

- **split input** into subgrids by stripping the partitions

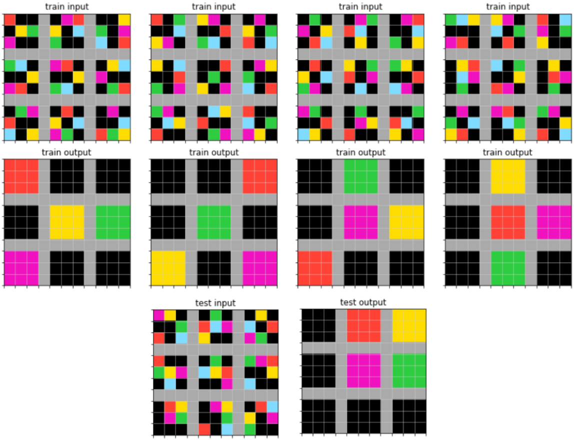
Example: ARC



High-level solution:

- **split input** into subgrids by stripping the partitions
- **find the special** subgrid that has only four non-blank cells

Example: ARC



High-level solution:

- **split input** into subgrids by stripping the partitions
- **find the special** subgrid that has only four non-blank cells
- **guess** an upscaled, grey-separated version of special subgrid

Example: ARC

- `SearchT` program that can solve both:

Example: ARC

- `SearchT` program that can solve both:

```
def abstactSolveSpecial = do
  inThings <- splitInputIntoThings
  (specialInput, in2outFn) <- synthAlignSpecialThingFn inThings
  pickSpecialFn <- synthPickSpecialFn specialInput inThings
  guess (in2outFn (pickSpecialFn inThings.test))
```

Example: ARC

- `SearchT` program that can solve both:

```
def abstactSolveSpecial = do
  inThings <- splitInputIntoThings
  (specialInput, in2outFn) <- synthAlignSpecialThingFn inThings
  pickSpecialFn <- synthPickSpecialFn specialInput inThings
  guess (in2outFn (pickSpecialFn inThings.test))
```

- A handful of `SearchT` tactics like this placed us in top 2%.
  - on ARC Kaggle competition
  - with no heuristics, only blind iterative-deepening
  - joint with: Ryan Krueger (UMich) and Jesse Michael Han (UPitt)

Example: ARC

- `SearchT` program that can solve both:

```
def abstactSolveSpecial = do
  inThings <- splitInputIntoThings
  (specialInput, in2outFn) <- synthAlignSpecialThingFn inThings
  pickSpecialFn <- synthPickSpecialFn specialInput inThings
  guess (in2outFn (pickSpecialFn inThings.test))
```

- A handful of `SearchT` tactics like this placed us in top 2%.
  - on ARC Kaggle competition
  - with no heuristics, only blind iterative-deepening
  - joint with: Ryan Krueger (UMich) and Jesse Michael Han (UPitt)
- Takeaway: programs + nondeterminism let you write:
  - convenient, abstract, compositional strategies
  - that solve superficially diverse problems

Example: ARC

- `SearchT` program that can solve both:

```
def abstactSolveSpecial = do
  inThings <- splitInputIntoThings
  (specialInput, in2outFn) <- synthAlignSpecialThingFn inThings
  pickSpecialFn <- synthPickSpecialFn specialInput inThings
  guess (in2outFn (pickSpecialFn inThings.test))
```

- A handful of `SearchT` tactics like this placed us in top 2%.
  - on ARC Kaggle competition
  - with no heuristics, only blind iterative-deepening
  - joint with: Ryan Krueger (UMich) and Jesse Michael Han (UPitt)
- Takeaway: programs + nondeterminism let you write:
  - convenient, abstract, compositional strategies
  - that solve superficially diverse problems
- Note: we needed to be conservative to keep search tractable.
  - could have written much more flexible tactics with good heuristics

Generic Heuristics

Generic Heuristics

- Recall a choice point consists of:
  - user-specificed data deemed relevant for decision
  - list of possible choices, each with summary data and a continuation

Generic Heuristics

- Recall a choice point consists of:
  - user-specificed data deemed relevant for decision
  - list of possible choices, each with summary data and a continuation

- But: the datatypes involved may be arbitrary.
  - inequalities: regular tactic state
  - Geometry: E-graph, sets for lines/circles, diagram
  - ARC: input and output grids

Generic Heuristics

- Recall a choice point consists of:
  - user-specificed data deemed relevant for decision
  - list of possible choices, each with summary data and a continuation
- But: the datatypes involved may be arbitrary.
  - inequalities: regular tactic state
  - Geometry: E-graph, sets for lines/circles, diagram
  - ARC: input and output grids
- In all three examples, subproblems see different data.
  - inequalities: `makeLookLike` sees the target pattern
  - Geometry: `chooseFromModel` sees desired property
  - ARC: `synthPickSpecialFn` sees labels, i.e. the special things

Generic Heuristics

- Recall a choice point consists of:
  - user-specficed data deemed relevant for decision
  - list of possible choices, each with summary data and a continuation
- But: the datatypes involved may be arbitrary.
  - inequalities: regular tactic state
  - Geometry: E-graph, sets for lines/circles, diagram
  - ARC: input and output grids
- In all three examples, subproblems see different data.
  - inequalities: `makeLookLike` sees the target pattern
  - Geometry: `chooseFromModel` sees desired property
  - ARC: `synthPickSpecialFn` sees labels, i.e. the special things
- Q: how to share statistical strength across all problems?

Compositional Embeddings

- First thought: emit tokens and use single transformer.
  - appealingly simple!
  - but: naïve, bad asymptotics, limited inductive bias

Compositional Embeddings

- First thought: emit tokens and use single transformer.
  - appealingly simple!
  - but: naïve, bad asymptotics, limited inductive bias
- Proposal: **compositional** embeddings.

Compositional Embeddings

- First thought: emit tokens and use single transformer.
  - appealingly simple!
  - but: naïve, bad asymptotics, limited inductive bias
- Proposal: **compositional** embeddings.
  - `List<`α`>` as (say) LSTM on embeddings of αs

Compositional Embeddings

- First thought: emit tokens and use single transformer.
  - appealingly simple!
  - but: naïve, bad asymptotics, limited inductive bias
- Proposal: **compositional** embeddings.
  - `List<`α`>` as (say) LSTM on embeddings of αs
  - `Vector<`α`>` as (say) transformer of αs

Compositional Embeddings

- First thought: emit tokens and use single transformer.
  - appealingly simple!
  - but: naïve, bad asymptotics, limited inductive bias
- Proposal: **compositional** embeddings.
  - `List<`α`>` as (say) LSTM on embeddings of αs
  - `Vector<`α`>` as (say) transformer of αs
  - `Set<`α`>` as AC-invariant repr of αs

Compositional Embeddings

- First thought: emit tokens and use single transformer.
  - appealingly simple!
  - but: naïve, bad asymptotics, limited inductive bias
- Proposal: **compositional** embeddings.
  - `List<`α`>` as (say) LSTM on embeddings of αs
  - `Vector<`α`>` as (say) transformer of αs
  - `Set<`α`>` as AC-invariant repr of αs
  - `Grid<`α`>` as CNN over αs

Compositional Embeddings

- First thought: emit tokens and use single transformer.
    - appealingly simple!
    - but: naïve, bad asymptotics, limited inductive bias
- Proposal: **compositional** embeddings.
    - `List<`α`>` as (say) LSTM on embeddings of αs
    - `Vector<`α`>` as (say) transformer of αs
    - `Set<`α`>` as AC-invariant repr of αs
    - `Grid<`α`>` as CNN over αs
    - custom `Term` type as GNN

Compositional Embeddings

- First thought: emit tokens and use single transformer.
  - appealingly simple!
  - but: naïve, bad asymptotics, limited inductive bias
- Proposal: **compositional** embeddings.
  - List<α> as (say) LSTM on embeddings of αs
  - Vector<α> as (say) transformer of αs
  - Set<α> as AC-invariant repr of αs
  - Grid<α> as CNN over αs
  - custom Term type as GNN
  - share *e.g.* Vector parameters across all vectors

Compositional Embeddings

- First thought: emit tokens and use single transformer.
  - appealingly simple!
  - but: naïve, bad asymptotics, limited inductive bias
- Proposal: **compositional** embeddings.
  - `List<`α`>` as (say) LSTM on embeddings of αs
  - `Vector<`α`>` as (say) transformer of αs
  - `Set<`α`>` as AC-invariant repr of αs
  - `Grid<`α`>` as CNN over αs
  - custom `Term` type as GNN
  - share *e.g.* Vector parameters across all vectors
  - (call this Phase I of the embedding)

Compositional Embeddings

- First thought: emit tokens and use single transformer.
  - appealingly simple!
  - but: naïve, bad asymptotics, limited inductive bias
- Proposal: **compositional** embeddings.
  - `List<α>` as (say) LSTM on embeddings of αs
  - `Vector<α>` as (say) transformer of αs
  - `Set<α>` as AC-invariant repr of αs
  - `Grid<α>` as CNN over αs
  - custom `Term` type as GNN
  - share *e.g.* Vector parameters across all vectors
  - (call this Phase I of the embedding)
- Embed the continuations as the program `Exprs`.

Compositional Embeddings

- First thought: emit tokens and use single transformer.
  - appealingly simple!
  - but: naïve, bad asymptotics, limited inductive bias
- Proposal: **compositional** embeddings.
  - `List<`α`>` as (say) LSTM on embeddings of αs
  - `Vector<`α`>` as (say) transformer of αs
  - `Set<`α`>` as AC-invariant repr of αs
  - `Grid<`α`>` as CNN over αs
  - custom `Term` type as GNN
  - share *e.g.* Vector parameters across all vectors
  - (call this Phase I of the embedding)
- Embed the continuations as the program `Exprs`.
- Phase II: after embedding all datatypes to same space:
  - run single generic model (*e.g.* transformer)
  - then at end, output floats giving scores to choices

- Now: we can embed arbitrary types into one space.
  - language features (*e.g.* typeclasses) make most plumbing transparent

Universal Oracle

- Now: we can embed arbitrary types into one space.
  - language features (*e.g.* typeclasses) make most plumbing transparent
- This lets us build:

Universal Oracle

A trainable procedure that can map any choice point with n choices encountered by any `SearchT` program into a vector of n floats, representing heuristic preferences among the choices.

Universal Oracle

- Now: we can embed arbitrary types into one space.
  - language features (*e.g.* typeclasses) make most plumbing transparent
- This lets us build:

Universal Oracle

A trainable procedure that can map any choice point with n choices encountered by any `SearchT` program into a vector of n floats, representing heuristic preferences among the choices.

- And finally we can implement:

Self-Improving Universal Search

A generic way of executing a `SearchT` program that queries the universal oracle at every choice point and trains the oracle based on new data arising from the search.

Outline

- Hypothesis:

  If we can win the IMO, we could use a similar methodology to automate any class of problems that

  - *are formally specified, and*
  - *that very smart humans can be trained to solve reliably.*

Beyond the IMO

- Hypothesis:

  If we can win the IMO, we could use a similar methodology to automate any class of problems that

  - *are formally specified, and*
  - *that very smart humans can be trained to solve reliably.*

- Does not include:
  - personal assistants (no formal spec)
  - solving Clay Millenium Problems (not trainable)

Beyond the IMO

- Hypothesis:

  If we can win the IMO, we could use a similar methodology to automate any class of problems that

  - *are formally specified, and*
  - *that very smart humans can be trained to solve reliably.*

- Does not include:
  - personal assistants (no formal spec)
  - solving Clay Millenium Problems (not trainable)

- But it is still is a huge class of important problems.

Beyond the IMO

- Includes most proofs in CS and Stats research.
  - convergence rates
  - regret/generalization bounds
  - asymptotic time/space arguments

Beyond the IMO

- Includes most proofs in CS and Stats research.
  - convergence rates
  - regret/generalization bounds
  - asymptotic time/space arguments

- Includes many subproblems that appear in “real” mathematics.
  - (many IMO problems arise this way)

Beyond the IMO

- Includes most proofs in CS and Stats research.
  - convergence rates
  - regret/generalization bounds
  - asymptotic time/space arguments

- Includes many subproblems that appear in “real” mathematics.
  - (many IMO problems arise this way)

- Includes big chunk of software verification!

Long-Term Aspiration

- **High-performance synthesis from extremely high-level code.**

Long-Term Aspiration

- **High-performance synthesis from extremely high-level code.**

- Universal finding: can be very difficult to write "good" specs.

Long-Term Aspiration

- **High-performance synthesis from extremely high-level code.**
- Universal finding: can be very difficult to write “good” specs.
- **But: it is always easier to write slow code than fast code!**

Long-Term Aspiration

- **High-performance synthesis from extremely high-level code.**
- Universal finding: can be very difficult to write “good” specs.
- **But: it is always easier to write slow code than fast code!**
- (Mostly) Well-defined and teachable to smart people:
  - start with: extremely high-level, naïve code
  - perform all sorts of high-, medium-, and low-level optimizations
  - end with: correct, super-high-performance code
  - (+ additional desiderata)

Long-Term Aspiration

- **High-performance synthesis from extremely high-level code.**
- Universal finding: can be very difficult to write "good" specs.
- **But: it is always easier to write slow code than fast code!**
- (Mostly) Well-defined and teachable to smart people:
    - start with: extremely high-level, naïve code
    - perform all sorts of high-, medium-, and low-level optimizations
    - end with: correct, super-high-performance code
    - (+ additional desiderata)
- If we can win the IMO, perhaps we can automate this too.

Thank You

- Lean4: `https://github.com/leanprover/lean4`
- IMO Grand Challenge website: `https://imo-grand-challenge.github.io/`
- Zulip channel: `https://leanprover.zulipchat.com/`

LEAN THEOREM PROVER

RiSE