Tentative Advice on Building w/ AI

Mike Dodds / mike@oath.tech

Oath Technologies

2026-08-21

How do we use AI to build formal tools?

Me, building formal tools

Some things I built (~2-3 months, *grind to victory* style)

- ACL2lean, a toolchain that replays ACL2 proofs in Lean
  - https://github.com/OathTech/ACL2Lean
- golean, a Go semantics and Iris-based formal verifier
  - https://github.com/OathTech/golean
- grossmith, a Go program generator (intended for differential testing)
  - https://github.com/OathTech/grossmith/
- brick-wp, a tactic library for BRiCk C++ verification
  - https://github.com/OathTech/brick-wp
- cerberus-lean, a port of Cerberus C semantics (+ Iris-based verification)
  - https://github.com/OathTech/cerberus-lean

(some other smaller things…)

Strategy: Grind to Victory

AI propensities (c. Fable / Sol - mid-2026)

AIs are good at:

- Math reasoning
- Coding / Lean
- Nit-picking
- Persistence!

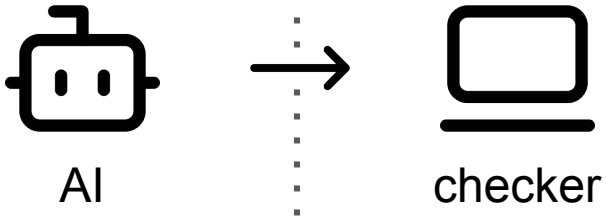
AIs are bad at:

- Judgement: *“am I doing the right thing?”*
- Creativity: *“is there some other way to solve this?”*
- Executive control: *“should I stop grinding and replan?”*

Untrusted search, trustworthy checks

Heuristic search
- Use any means available
- Quick, powerful, opaque
- May be wrong

Verification
- Highly predictable methods
- Slow, simple, legible
- Guaranteed correctness



Untrusted **Trusted**

Grind to victory

Ingredients

- Objective: something you want to build
- Oracle: a checkable root of trust that points towards the objective

Process:

1. AI agent builds some stuff
2. AI agent checks oracle, fixes gaps

(iterate for a long time until success)

Oracles

- Tests / differential testing
- Existing implementation
- Theorem
- Coverage metric
- Agent judge / human review …

The important thing:

- Oracle is hard to game
- Oracle can be human reviewed
- Oracle points to objective

Building a Go verifier (1): testset

Build a testset that cover a lot of Go features

Oracle:

- The test set compiles and runs
- Test internal features affect return values

Grind to victory!

Tool: https://github.com/OathTech/golean

Building a Go verifier (2): Go semantics

Build a Lean interpreter that exactly matches Go

Oracle:

- Every conformance test runs in the interpreter
- Differential testing: tests produce the same result on real-Go and Lean

Grind to victory!

Building a Go verifier (3): semantics engineering

Build relational semantics which is compatible with iris-lean

Oracle:

- Lean interpreter and relational semantics are proved equivalent
- Iris properties exist over the relational semantics

Grind to victory!

Building a Go verifier (4): verifying code

Verify small Go programs

Oracle:

- Theorems exist about a set of target programs
- The TCB excludes the relational semantics

Grind to victory!

Building a Go verifier (5): example generation

Automatically generate many small Go programs

Oracle:

- The generator always generates conformant Go
- The generator covers some set of Go features

Grind to victory!

Fuzzer: https://github.com/OathTech/grossmith/settings

ACL2lean: https://github.com/OathTech/ACL2Lean

- ACL2 doesn't have proof objects
- Very different proof model to Lean
- Can we replay ACL2 proofs in Lean?

Requires the AI to build:

- Deep embedding of ACL2 logic in Lean
- Instrumentation of ACL2 tool to emit proof logs
- Tactics engineering to replay proofs

(Very difficult, never done before - 1+ research papers, maybe whole PhD)

ACL2Lean grind loops

Many grind loops across the project:

- Instrument ACL2 to generate proof logs (grind on coverage)
- Parse proof logs into trees (agent-judge on small examples)
- Check trees are valid reasoning (grind on coverage)
- Replay trees as tactics (grind on kernel pass)
- …

Start with thin slices, pry the window wider

- Support one simple ACL2 proof (theorem: list ++ list = list or similar)
- *Grind to victory*
- Add one more category of proof
- *Grind to victory*
- *…*
- ACL2Lean support a real ACL2 book! ← **today**

Widen the window by a sequence of grind campaign, *each* with goals / oracles

Future targets for *grind to victory* campaigns

- SpecTec in Lean (trash prototype Claude built overnight: https://github.com/OathTech/spectec-lean)
- etcd-io/raft (via golean)
- libxml2 (via cerberus-lean)
- Verifying the Rust compiler
- Verifying the Linux kernel :)

Other advice

Don’t read the code

At least most code

Your most precious resource is *time* - oversight and understanding is expensive!

You need an anchor objective

Some top-level auditable claim you understand e.g

$$\forall\ x \in \text{args},\ \forall\ r,$$

$$\text{goSemantics}\ p\ x = \text{some}\ r \quad \rightarrow \quad r = .\text{ok}\ (\text{specFn}\ x)$$

Otherwise you might build total nonsense!

Use the best model

Current best models (c. August 21 2026)

- Claude Fable
- GTP-5.6 Sol

Other models are much worse especially for conceptual / long cycle work

When the next model arrives, pick it up and test it

Pay for AI

- $200 / month Claude plan is a great deal - *much* cheaper than API costs
- Similar for Codex

Typical buildout for a useful formal thing will be ~$1000s of tokens

Work in arcs

- Describe a goal - a single grind campaign
- Agent defines an ‘arc charter’ - a doc in the repo

Use /goal - forces the agent to continue until a condition is done

MY GRIND PROMPT (as of 2026-08-21 - messy, work in progress)

/goal Finish ALL of the work chartered in **[charter file]**

We are going to do a fully autonomous push on some work. You are a long-running autonomous agent and your job is to complete the WHOLE remainder of this work as defined in the charter file. Completion means ALL the work is done, COMPLETELY. Working autonomously means the agent (rather than the user) will make many judgement calls. Judgement calls typically do NOT need to be checked with the user. Resolve them according to long-term project principles.

All decisions should be logged for later review, but during this period, making calls is for YOU, the agent. Don't stop to solicit feedback from the user, make the call and keep going (the exception is a true emergency, see below). Decisions must be clearly logged, and it is critically important that agent-decided decisions are logged as such (not confused with user-decided decisions). You should be very careful to tag any decisions along the way with [AGENT] or [USER] . It would be a critical trust failure for an agent-decided decision to be mistakenly logged as user-decided.

Work on a branch and create sub-branches if you need them. Do not merge to main - that will happen once the goal is done and you have user sign-off.

EMERGENCY EXIT CONDITION: The agent may declare an EMERGENCY EARLY EXIT from the goal. The agent should declare the NATURE of the emergency as part of declaring the emergency exit. However, an emergency exit will ALWAYS be permitted without question no matter the reason given. This is to allow the agent a completely reliable escape hatch in unanticipated situations.

The agent should exercise judgement and only call for an emergency exit in situations where the work is truly stuck (e.g. blocked by lack of resources, uncompletable thanks to mis-specification, spinning with minimal progress), or when facing a dire threat to project success, or other true emergencies. The agents should not declare an emergency exit for routine decisions. When faced with decisions, the agent should make the decision and log it for later review (see above).

Adversarial audit

Agents are good at the *goal they are set*

Dispatch review agents at the end of each arc

- Multiple subagents
- Brief the agents to be adversarial / decorrelated

Very effective at killing bugs!

Run in a sandbox

- Stops the agent from accessing the network (except AI API)
- Stops the agent accessing anything outside current folder and allow-list

I like https://nono.sh/

Agents will break things if you don’t restrict them

Try to scale up

Human time / oversight is the most punishing bottleneck

Scale up hierarchy:

- *[less scalable]* Human reviews every action / edit
- Human reviews every change at checkpoints
- Human discusses each commit w/ agent
- Human reviews arc plans, human watches commits go by
- Multiple arcs run at once, human+agent management ← **me today**
- *[more scalable]* Agents decides arcs, agents supervise agents. Human sets working practices

Be insanely ambitious

- Most people are aiming too low. Hard tasks reveal the boundary
- Calibration on capabilities is very difficult

Eg. these tasks will fail today, but it's interesting *how* they fail

- "Verify linux kernel"
- "Verify rustc"

Try to figure out what would make such tasks successful

Be ready for the next model

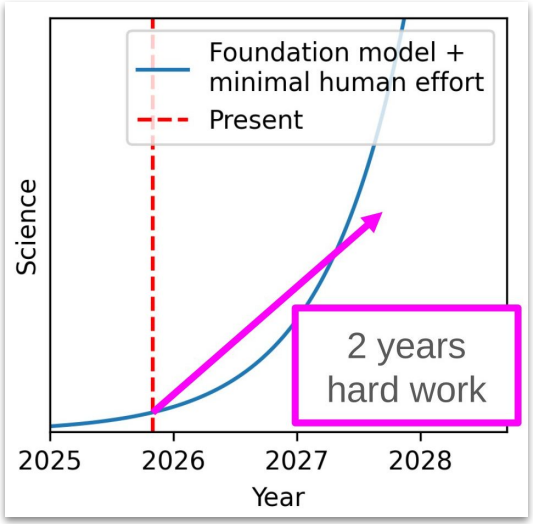
Build for next year

Not today / yesterday:

Common failure:

- “AIs are bad at *X*. I will build a tool that solves *X*”
- *… 6 months pass*
- AIs are good at X :(

Not for +10 years:



“Advice for a (young) investigator in the first and last days of the Anthropocene”
Jascha Sohl-Dickstein
https://iclr.cc/virtual/2026/10022182

Mike Dodds / mike@oath.tech

Oath Technologies

2026-08-21