
Neuro-symbolic Requirements Elicitation

Utilizing Formal Verification Counterexamples as Contextual Prompts

Okba Tbermacine

ENSIA, Algiers, Algeria

Chouki Tbermacine

IRISA, Univ. Southern Brittany, Vannes, France

41st IEEE/ACM International Conference on Automated Software Engineering (ASE) 2026, Munich, Germany



Outline

1. Context & Problem
2. The C-GRiD Approach
3. Evaluation
4. Discussion & Conclusion

Motivation: The Specification Gap

- ▶ Many failures in complex systems stem not from coding bugs but from **incomplete, ambiguous or contradictory requirements**
- ▶ Domain experts hold a **latent mental model**: unstated assumptions, unexplored edge cases

Formal Methods (TLA+, Alloy)

- + Mathematical rigor, correctness guarantees
- Steep learning curve $\Rightarrow$ **Blank Page Problem**: experts rarely write or read formal specs

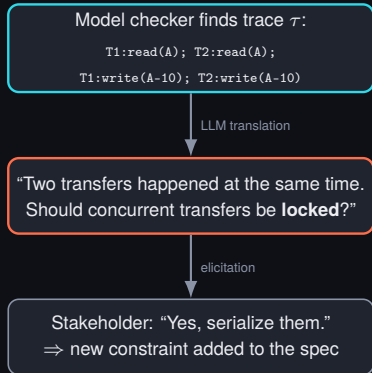
Large Language Models

- + Natural-language fluency, informal $\rightarrow$ formal translation
- No logical grounding $\Rightarrow$ **Logical Grounding Problem**: hallucinated constraints, missed corner cases

How do we turn informal stakeholder intent into a **complete, formally verifiable** specification?

Core Insight: Counterexamples as Prompts

- ▶ A **counterexample** is a finite execution trace witnessing a property violation
- ▶ Usually a verdict on a *final* spec. . .
- ▶ . . . we use it as a **deterministic, semantic prompt** signalling a missing or incorrect requirement
- ▶ Formal verification becomes an **interactive discovery engine**, not a terminal correctness check



Positioning

Paradigm	Refines	Inputs $\rightarrow$ Outputs	Representative works
Translation	Properties Φ	NL $\rightarrow$ LTL/CTL formulas	ARSENAL, NL2LTL, NL2CTL
Validation	Formulas Φ	NL $\rightarrow$ candidate formulas $\rightarrow$ user validation via traces	LTLTalk, SynthTL, ARTEMIS
Joint elicitation	Model δ and Φ	Incomplete spec $\rightarrow$ verified LTS	C-GRID (this work)

vs. AI-for-RE & formalisation

Existing work stays at the **informal-text level** or translates **already-stated** requirements; neuro-symbolic tools use counterexamples to *repair* artefacts against a fixed baseline

Not CEGIS

CEGIS¹ refines a *candidate program* against a fixed spec, with a solver as oracle. C-GRID refines the **problem statement itself** (Φ, δ), with a human stakeholder as oracle.

¹CEGIS: Counterexample-Guided Inductive Synthesis. A. Solar-Lezama, *Program Synthesis by Sketching*, PhD thesis, UC Berkeley, 2008.

Formal Foundations

The system is modelled as a **Labeled Transition System**:

$$\mathcal{M} = \langle \Sigma, \Sigma_0, \mathcal{A}, \delta, \Phi \rangle$$

- ▶ Σ states (valuations of variables V), $\Sigma_0 \subseteq \Sigma$ initial states, $\mathcal{A}$ actions
- ▶ $\delta : \Sigma \times \mathcal{A} \rightarrow \Sigma$ **system model**; Φ **requirements properties** in LTL (safety $\Box P$, liveness $\Diamond Q$)

Both start imperfect

$$\Phi_{init} \subset \Phi_{ideal} \quad \delta_{init} \neq \delta_{ideal}$$

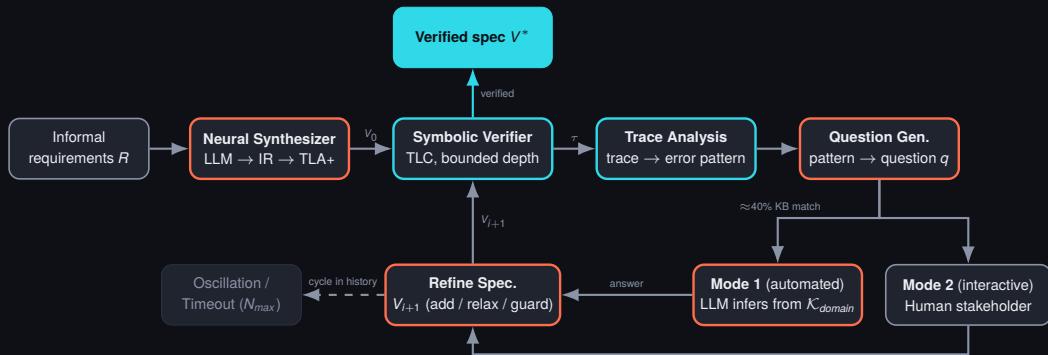
ideal = latent stakeholder intent

Goal

Discover $\Phi_{ideal} \setminus \Phi_{init}$ by treating property violations as **probes into the stakeholder's mental model**

Scope: behavioural temporal properties over discrete LTSs; performance, usability, etc. are out of scope.

The C-GRiD Loop



Neural components (LLM) || **Symbolic** components (sound, deterministic) — constrained decoding for V_0 only

From Counterexample to Question

Trace Analysis (purely symbolic)

1. Locate the shortest violating prefix $\tau[0..i]$
2. Backtrace the minimal causal actions σ
3. Classify the error pattern:

Pattern	Question template
MissingGuard	"Should <i>action</i> be allowed when <i>cond</i> ?"
RaceCondition	"Should concurrent <i>actions</i> be serialized?"
Deadlock / Logic	No template: the LLM writes the question

Question Generation (neuro-symbolic)

- ▶ Domain ontology maps variables to business concepts
- ▶ Templates per pattern: **predictable**, **hallucination-resistant** questions

Logistics — raw trace:

State 5: Ord.st=Ship; Action: Cancel; Result:
Ord.st=Can

⇓ generated question

"The system allowed an order to be cancelled after it was already marked as *Shipped*. **Is this transition allowed?**"

Symmetric Resolution & Convergence

A counterexample on V_0 may reveal a **missing constraint** or an **over-constrained translation** — the **polarity of the answer** decides the repair; the stakeholder never reasons about logic.

Answer A: “This is allowed” $\Rightarrow$ relax

Repair: weaken a property in Φ

BasicTransfer: LLM produced

$\Box(balance \geq 100)$. A balance of 50 is flagged;

answer: *any non-negative balance is fine*

$\rightarrow \Box(balance \geq 0)$

Answer B: “This is disallowed” $\Rightarrow$ restrict

Repair: new property in Φ or stronger guard in δ

MutualExclusion: $\Box\neg(active_1 \wedge active_2)$ violated

by a concurrent trace; answer: *serialize*

$\rightarrow$ mutex guard added to δ

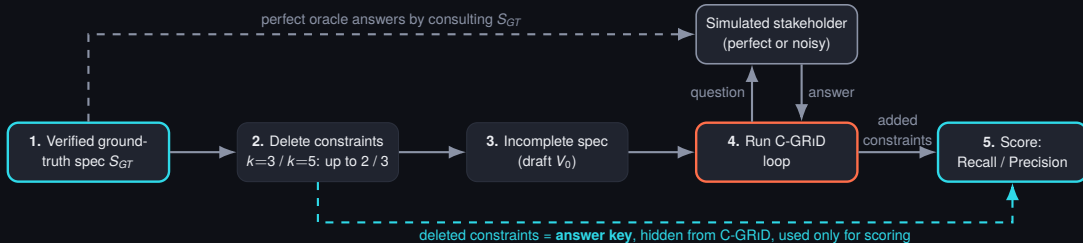
Does the loop terminate?

Perfect oracle: yes. Each “disallowed” answer removes at least one behaviour from a finite state space, and each property is relaxed at most once.

Noisy oracle: not guaranteed. It stops on oscillation (a repeated spec) or after N_{max} iterations.

Evaluation Design: Constraint-Removal Protocol

Requirements that are **unknown** can't be counted $\Rightarrow$ hide **known** ones, then check what comes back.



RQ1 — Effectiveness

Does it recover missing constraints better than the baselines?

Varies: **the approach**

RQ2 — Robustness

Does it still work when the stakeholder gives wrong answers?

Varies: **the oracle**

RQ3 — Scalability

How does it cope with more gaps and larger specs, at what cost?

Varies: **the problem**

Same protocol, one factor at a time: **ideal conditions** $\rightarrow$ **imperfect human** $\rightarrow$ **harder problems**

Benchmark & Setup

10 verified specifications (literature)

Specification	Domain	Vars	Constr.
BasicTransfer	Banking	3	4
ShoppingCart	E-commerce	4	5
OverdraftProtection	Banking	4	6
PatientRecords	Healthcare	5	5
MutualExclusion	Concurrency	3	4
SpecificationName	Generic	2	2
TwoPhaseCommit	Distributed	8	10
AuctionProtocol	Distributed	6	9
ElevatorController	Embedded	10	15
TrafficLightController	Embedded	7	11

Setup

- ▶ `llama-3.3-70b-versatile` (Groq), Temp 0.3
(model-agnostic loop; Temp: reproducible yet diverse)
- ▶ TLC, depth 15, $N_{max}=10$, 300 s timeout
(bounded depth: fast feedback, as the loop is interactive)

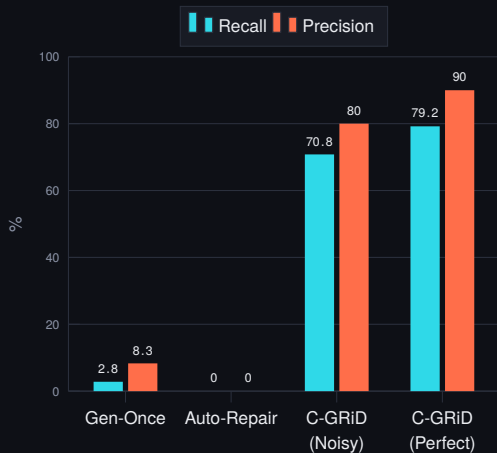
Simulated oracles

- ▶ **Perfect**: consults S_{GT} (upper bound)
- ▶ **Noisy**: 75% correct, 15% ambiguous, 10% wrong

Baselines

- ▶ **Gen-Once**: single-pass LLM, few-shot prompt
- ▶ **Auto-Repair**: CEGIS-style, same trace analysis + fixed syntactic repair templates

RQ1 — Effectiveness



- ▶ Perfect oracle: **79.2% recall, 90.0% precision** (F1 0.83)
- ▶ Gen-Once: 2.8% — single-pass generation rarely uncovers a missing requirement
- ▶ Auto-Repair: 0% — syntactic templates cannot capture domain semantics

Paired test (6-spec subset, 12 configs)

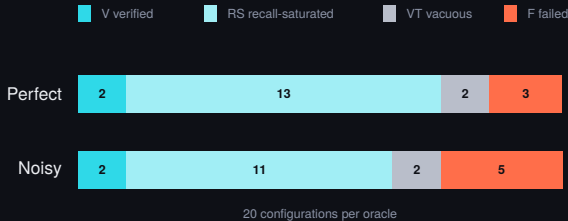
65.3% vs. 2.8% recall: **+62.5 pp**

95% CI [34.6, 90.4], $t(11)=4.93$, $p<.001$

Cohen's $d_z=1.42$: **very large**

Cohen's d : 0.2 small, 0.5 medium, 0.8 large

How Runs Terminate & RQ2 — Robustness to Noise



Takeaway: finding $\neq$ integrating

Dominant outcome is **RS (Recall-Saturated)**: all removed constraints found, but no clean re-verification within N_{max} . C-GRID reliably *finds* missing constraints; reliably *integrating* them is still open.

RQ2: 25% faulty answers

- ▶ **Recall** 79.2% $\rightarrow$ **70.8%**: only 8.4 points lost
- ▶ **Runs that did not fail**: 17/20 $\rightarrow$ 15/20
- ▶ Bad answers are often **caught**: the next model check exposes them and triggers a new question
- ▶ Noisy runs look shorter (6.3 vs. 7.1 iterations) only because failed runs stop early

RQ3 — Efficiency & Scalability

By incompleteness level (perfect oracle)

	$k = 3$ ≤ 2 removed	$k = 5$ ≤ 3 removed
Recall	85.0%	73.3%
Precision	90.0%	90.0%
Runs not failed	10/10	7/10

- ▶ More gaps $\Rightarrow$ **graceful degradation**, no collapse
- ▶ Precision unchanged: what is added stays mostly correct
- ▶ ≈ 7 iterations on average (max. 10)

What do the numbers *not* show?

3 largest specs: only 1 **constraint** removed
 $\Rightarrow$ 100% recall there is the easy case. Size does not destabilise the loop, but larger gaps are untested.

No counterexample, no question: in OverdraftProtection ($k=3$), the removed constraints were **redundant** or **not checked by any property**
 $\Rightarrow$ nothing to ask, yet the run counts as “not failed”.

Failure Analysis & Threats to Validity

Where does it fail?

- 1. Asking works.** Even the 3 failed runs recovered correct constraints (2 of 3 each).
- 2. Editing breaks.** When the LLM inserts a correct constraint into the TLA+ spec, it sometimes **breaks another part** → new counterexample → the loop chases its own errors. Same cause behind the 13 recall-saturated runs.
- 3. Planned fix:** (not yet evaluated) check every edit before accepting it.
Re-run the model checker on the properties that *already held*. If one now fails, the edit broke something → **discard it** and ask the LLM to redo it (max. 3 tries).

Threats

- ▶ **Construct:** simulated oracles $\neq$ real stakeholders (who change their minds); recall counts only constraints expressible in the model's variables
- ▶ **Internal:** LLM non-determinism (mitigated: fixed seed, archived runs)
- ▶ **External:** ≤ 10 vars / 15 constraints, single LLM; 50+ variables unvalidated — compositional verification is a *prerequisite* for industry
- ▶ **Conclusion:** small samples (20 / 12 configs)

Conclusion & Future Work

Summary

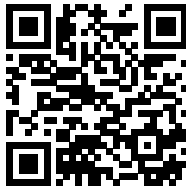
C-GRID turns model-checking counterexamples into **questions for stakeholders**.

✓ Finds missing constraints: **79%** recall, **71%** with 25% faulty answers; baselines $\leq 3\%$

✗ Still open: integrating them **without breaking the spec**

Future work

- ▶ **Safer edits**: check each edit; only add restrictions
- ▶ **Real users**: studies with engineers & domain experts
- ▶ **Scale**: verify large specs module by module
- ▶ **Broader inputs**: diagrams, logs



Replication package:
benchmarks, prompts, results
& code

`doi.org/10.5281/zenodo.19222714`

Thank You

Questions?

Okba Tibermacine

`okba.tibermacine@ensia.edu.dz`

Chouki Tibermacine

`chouki.tibermacine@univ-ubs.fr`

Replication package: doi.org/10.5281/zenodo.19222714

