Event-B Agent: Towards LLM Agent for Formal Model Synthesis and Repair

Hongshu Wang[1], Xinyue Zuo[1], Yuhan Sun[2], Qin Li[2], Yamine Ait Ameur[3], Jin Song Dong[1]

1 National University of Singapore, 2 East China Normal University, 3 IRIT – National Polytechnic Institute of Toulouse

Background

Formal Methods

- **Formal methods** use mathematical models and proofs to provide stronger correctness guarantees than testing alone.
- This is increasingly important for **LLM-generated software**, where subtle errors may be difficult to detect through testing.
- **Correct-by-construction** development incrementally refines requirements into implementations and formally verifies correctness at each step.
- This procedure remains **manual, time-consuming, and expertise-intensive**: even moderately complex models can generate hundreds of proof obligations.

Autoformalization

- LLMs show promise for automating formal methods, but existing work targets **isolated tasks**, e.g., specification synthesis, theorem proving, or model generation.
- Model checking provides **bounded verification**, neglecting theorem proving and refinement which can establish **unbounded correctness guarantees**.
- The key challenge is that **models and proofs must evolve together**: a failed proof may require repairing the model, the proof, or both.
- **Gap:** Existing approaches do not systematically support this joint, iterative **model synthesis and proof repair** process.

Methodology

To address the above mentioned challenge, we propose Event-B Agent, an LLM-powered end-to-end framework for formal model synthesis and repair. Figure 1 shows the overall workflow of Event-B Agent, and Figure 2 illustrates the minimum-searching model generated by Event-B Agent.

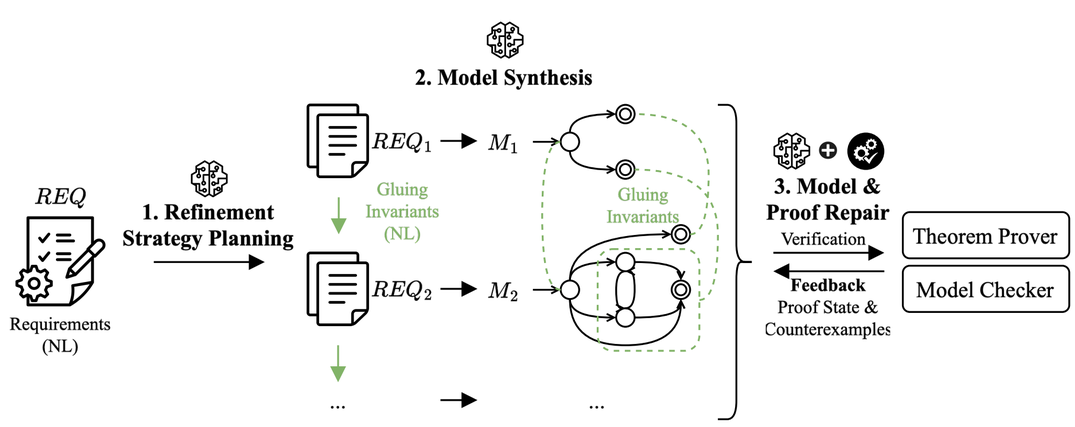

Figure 1. Overview of Event-B Agent. From natural-language requirements (REQ), the system plans a refinement strategy to guide incremental model synthesis, with verification and repair at each refinement level.

1. Refinement Strategy Planning

Given a natural-language system description, Event-B Agent generates a **refinement strategy** that distributes requirements across refinement steps and summarizes the **gluing invariants** between them.

For the minimum-searching example (Table 1), the agent generates two refinement steps: $REQ_{M_1} = \{FUN\text{-}1\}$ and $REQ_{M_2} = \{FUN\text{-}2, \ldots, FUN\text{-}6\}$. It also generates the gluing invariant $\mathcal{I}_{g_2} = \{inv_{g_2}\}$ to relate the two steps (Figure 2).

2. Model Synthesis

For each refinement step i, Event-B Agent synthesizes a **formal model** from $REQ_{M_i} \cup \mathcal{I}_{g_i}$. Figure 2 shows the generated $M_{abstract}$ and $M_{concrete}$, where edges represent **events** with guards and actions. Through refinement, properties proven in the abstract model are **preserved** in the concrete model without being re-proven.

Table 1. Equipment (EQP) and Function (FUN) requirements of minimum-searching model.

ID	Description
EQP-1	Provides a finite natural number function f defined on the domain $\{0..n-1\}$.
EQP-2	Provides n as the size of the domain of f.
FUN-1	The final condition requires $j \in \text{dom}(f)$ and $f(j) = \min(\text{ran}(f))$.
FUN-2	The system has a boolean variable `searching` to indicate whether the minimum has been fully identified. Two indices are maintained: i as the scanning index and j as the current candidate for the minimum.
FUN-3	Initialization sets i = 1 and j = 0, with searching = true.
FUN-4	While `searching` is true, if $i \neq n$ and $f(i) < f(j)$, update $j := i$ and increment i.
FUN-5	While `searching` is true, if $i \neq n$ and $f(i) \geq f(j)$, keep j unchanged and increment i.
FUN-6	When $i = n$, `searching` becomes false, marking the completion of searching.

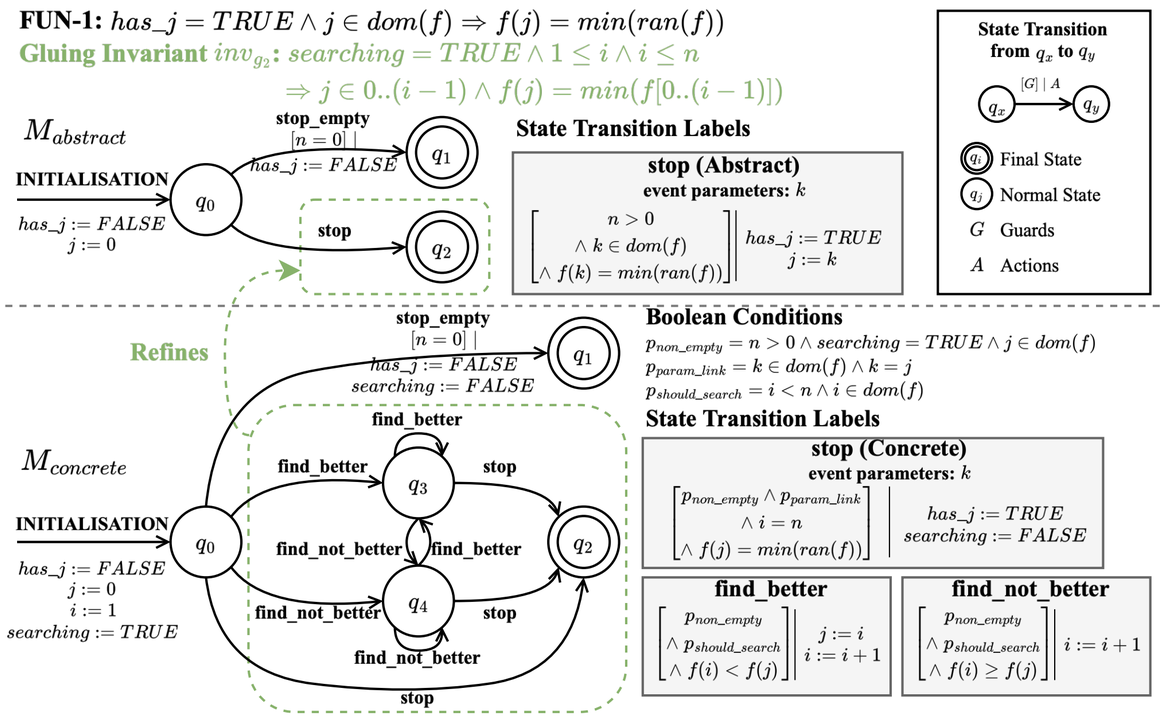

Figure 2. The abstract and concrete formal models constructed by Event-B Agent for the motivating example.

3. Model & Proof Repair

Each model is verified through **model checking and theorem proving**, with LLM-proposed repairs applied to the **model, proofs, or both** (Figure 4). This process repeats until all proof obligations are discharged or the trial limit is reached.

Evaluation

Baselines

1. **LLM + auto provers.** This baseline generates specifications with an LLM and discharges the resulting proof obligations using automated provers.
2. **Cursor.** Cursor is a commercial coding LLM agent that is adapted as a baseline.
3. **PAT-Agent.** PAT-Agent is one of the first formal LLM agents, and it uses model checking as feedback to iteratively guide the model construction.

Metrics

For a model M with a set of proof obligations PO_M, we use Proof Obligation Discharge Rate (PDR), Requirement Coverage (RC) and Fulfilment (RF) Rates as the evaluation metrics:

$$PDR = \frac{\sum_{po \in PO_M} \mathbb{I}[Discharged(M, po)]}{|PO_M|}, \quad RC = \frac{\sum_{req \in REQ_M} \mathbb{I}[Covered(M, req)]}{|REQ_M|},$$

$$RF = \frac{\sum_{req \in REQ_M} \mathbb{I}[Covered(M, req) \wedge (\bigwedge_{po \in PO_{req}} Discharged(M, po))]}{|REQ_M|}$$

Overall Performance

Table 2. Overall performance. Event-B Agent consistently outperforms the three baselines on the metrics PDR, RC, and RF. GPT-5 with medium reasoning level is used as the backbond LLM for all methods.

Method	PDR	RC	RF
LLM + auto provers	0.8920	0.9250	0.6896
Cursor	0.9007	0.8928	0.6886
PAT-Agent	0.9556	0.9205	0.7578
Event-B Agent	**0.9786**	**0.9713**	**0.9379**

How does Event-B Agent construct and repair models?

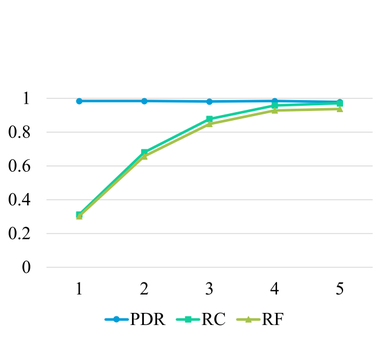

Figure 3. Evolution of the three metrics across refinement steps.

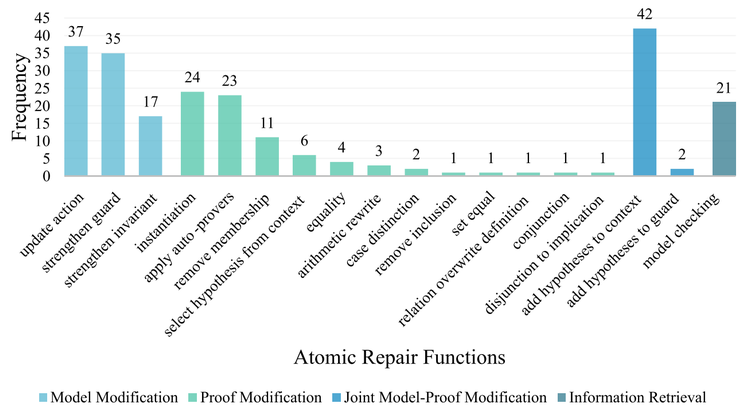

Figure 4. Distribution of model and proof repair functions contributing to successful PO discharge.

Paper: https://dl.acm.org/doi/abs/10.1145/3808218

Github: https://github.com/HongshuW/EventB_Agent

hongshu.wang@u.nus.edu