MANTLE: A TYPED SAFETY IR FOR TOOL AGENTS UNDER INTERFACE DRIFT

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-agent safety logic is commonly maintained independently in adversarial discovery, trace repair, risk prediction, and runtime enforcement. A changed interface can therefore leave four consumers attached to different meanings of the same hazardous action. We introduce Mantle, a typed safety intermediate representation (Safety IR) that assigns a stable, versioned identity to each hazard predicate and compiles that identity into the four lifecycle consumers. Mantle makes a semantic update an auditable systems operation: a validated predicate revision relabels prediction targets, refreshes repair constraints, updates exploration rewards, and changes the runtime action rule together. We also formalize a maintenance frontier that compares shared and stage-local designs under fixed model, data, compute, and human-maintenance budgets. The formulation separates candidate selection from held-out paired assessment and states the matching, denominator, and uncertainty contracts needed to identify cross-stage propagation. This yields a concrete design criterion for tool-augmented agents: safety maintenance should be evaluated by the complete effect of a semantic edit across the lifecycle, not by the behavior of one isolated guard.

1 INTRODUCTION

Tool agents turn model decisions into persistent state changes, including repository edits, database transactions, credential transfers, and external commitments (Yao et al., 2023; Schick et al., 2023; Qin et al., 2024; Liu et al., 2024; Zhou et al., 2024; Jimenez et al., 2024). Their safety logic spans a lifecycle. Red-team discovery identifies dangerous traces, repair search constructs safe continuations, predictors estimate imminent risk, and runtime guards decide whether to continue. When an API is renamed or an argument acquires a new side effect, each stage needs the same updated semantic interpretation.

Many existing safety approaches focus on individual lifecycle stages, including high-assurance specification and verification (Dalrymple et al., 2024), rule-based rewards (Mu et al., 2024), red-team discovery (Perez et al., 2022; Ganguli et al., 2022; Ge et al., 2023), and verbal feedback (Shinn et al., 2023); see Ji et al. (2023) for a broader overview. This leaves a systems question: what is the maintained object that guarantees these consumers continue to agree after an interface changes? Mantle answers with a typed Safety IR. A versioned predicate identity is the canonical semantic object; deterministic projections compile it into exploration reward, repair constraints, prediction targets, and runtime intervention.

The representation matters because it makes cross-stage propagation both inspectable and comparable. A semantic revision has a single version record, four compiled consumers, and an explicit transition rule. A stage-local control instead maintains four surfaces separately. Holding models, data, compute, evaluation access, and human-maintenance budgets fixed then isolates the consequence of maintaining one shared semantic identity.

We make four contributions:

- a versioned typed Safety IR whose predicate records carry the operation signature, semantic predicate, runtime guard, and escalation policy required by all four lifecycle projections;

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

- a deterministic update and runtime semantics that specifies how a predicate change relabels prediction targets, refreshes repair constraints, and selects execution, repair, block, or escalation;
- a finite-candidate maintenance frontier that separates calibration-time selection from held-out paired assessment; and
- a matched evaluation contract covering stage-local controls, binary metric denominators, cluster structure, external-defense selection, and derivative-suite construction.

2 Problem Formulation

We consider a tool-augmented agent that operates over a typed interface. At step t, the agent receives a task and an execution history, selects an operation and typed arguments, observes the return value and state transition, and continues until termination.

Definition 1 (Typed tool trace) *A typed tool event is $e_t = (\mathrm{op}_t, x_t, r_t, s_t, s_{t+1})$, where op_t is an operation, x_t is an argument satisfying that operation's type signature, r_t is its return value, and s_t, s_{t+1} are the pre- and post-states. A trace $\tau = (e_1, \ldots, e_T)$ is the ordered execution history; let $\mathcal{T}$ denote the set of finite typed tool traces.*

Definition 2 (Versioned hazard predicate) *A predicate record is*

$$p_k^v = (\mathrm{id}_k, v, \mathsf{Sig}_k, \mathsf{Sem}_k, \mathsf{Guard}_k, \mathsf{Esc}_k),$$

where id_k is a stable identity, v is a monotonically increasing version, Sig_k is the typed operation signature, $\mathsf{Sem}_k : \mathcal{T} \to \{0,1\}$ determines whether a trace contains the hazard, Guard_k specifies an immediate hard guard, and $\mathsf{Esc}_k \in \{0,1\}$ determines whether an unrepaired trigger requires human escalation. A hazard grammar is a finite set $\mathcal{H}^v = \{p_1^v, \ldots, p_K^v\}$. A predicate revision preserves id_k and creates a new version record; a semantic split or merge creates an explicit lineage relation in the registry.

The grammar contains families for destructive writes, state corruption, credential exposure, irreversible effects, and authorization escalation. A family is a semantic category; a concrete predicate record binds the family to typed operations and state conditions in one interface. This distinction makes the object being edited unambiguous.

Definition 3 (Lifecycle Safety IR) *Let C_q be a deterministic compiler for consumer $q \in \{\mathrm{exp}, \mathrm{rep}, \mathrm{pred}, \mathrm{run}\}$. The Safety IR projection tuple is*

$$\Gamma(\mathcal{H}^v) = (C_{\mathrm{exp}}(\mathcal{H}^v), C_{\mathrm{rep}}(\mathcal{H}^v), C_{\mathrm{pred}}(\mathcal{H}^v), C_{\mathrm{run}}(\mathcal{H}^v)).$$

The four compiled objects respectively define witness rewards, admissible repair continuations, partial-trace target labels, and runtime actions. An update is coherent when every compiler consumes the same committed grammar version and every compiled output carries that version identifier.

Definition 4 (Maintenance frontier) *For architecture $c \in \{\mathrm{IR}, \mathrm{local}\}$ and budget $b = (b_e, b_h)$, let $\mathcal{A}_c(b)$ be a finite, predeclared candidate set of adaptations whose edit count and recorded engineering time do not exceed b_e, b_h. Let $\widehat{R}_{\mathrm{cal}}$ and $\widehat{M}_{\mathrm{cal}}$ be calibration estimates of catastrophic risk and mission completion, and let m_0 be a fixed completion floor. The selected candidate is*

$$\theta_c^\star(b) = \operatorname*{arg\,min}_{\theta \in \mathcal{A}_c(b) : \widehat{M}_{\mathrm{cal}}(\theta) \geq m_0} \left(\widehat{R}_{\mathrm{cal}}(\theta), -\widehat{M}_{\mathrm{cal}}(\theta), E(\theta), H(\theta), \mathrm{hash}(\theta) \right),$$

where the tuple is ordered lexicographically, E is the concrete-predicate edit count, and H is engineering time. The final component resolves all remaining ties deterministically. With a held-out test risk R_{test}, reference θ_{ref}, and no use of held-out test outcomes in selection, the maintenance frontier is

$$\mathcal{F}_c(b; m_0) = R_{\mathrm{test}}(\theta_{\mathrm{ref}}) - R_{\mathrm{test}}(\theta_c^\star(b)), \qquad \mathcal{A}_{\mathrm{prop}}(b; m_0) = \mathcal{F}_{\mathrm{IR}}(b; m_0) - \mathcal{F}_{\mathrm{local}}(b; m_0).$$

This definition characterizes an estimand rather than a population maximum over unobserved operating points. It fixes the candidate set, feasibility condition, selection order, and comparator before paired held-out assessment.

3 MANTLE: ONE SEMANTIC OBJECT, FOUR LIFECYCLE CONSUMERS

Mantle uses the predicate registry as the maintained object. A committed update is accepted only if its typed signature parses, its lineage is well formed, its semantic checker is total on the declared event schema, and its runtime requirements are present. The compiler then emits four version-stamped outputs. The explorer receives family-balanced witness rewards; repair search receives a predicate-safe admissibility test; the forecaster receives predicate-indexed labels; and the controller receives the hard guards, threshold, and escalation policy. A stage-local implementation must reproduce these outputs with separately maintained surfaces.

3.1 SEMANTIC UPDATE RULE

For a revision $\mathcal{H}^v \to \mathcal{H}^{v+1}$, Mantle first constructs a typed migration map from every affected old operation binding to its new binding. A record with an unresolved binding fails validation and is not committed. For a committed revision, the update transaction is

$$\begin{aligned} \mathcal{H}^{v+1} &\leftarrow \mathsf{ValidateAndCommit}(\mathcal{H}^v, \Delta), \\ (\rho_{\mathrm{exp}}, \rho_{\mathrm{rep}}, \rho_{\mathrm{pred}}, \rho_{\mathrm{run}}) &\leftarrow \Gamma(\mathcal{H}^{v+1}), \\ \mathcal{B}^{v+1} &\leftarrow \mathsf{Relabel}(\mathcal{B}^v, \mathcal{H}^{v+1}), \qquad f^{v+1} \leftarrow \mathsf{Invalidate}(f^v, \mathcal{B}^{v+1}). \end{aligned}$$

Here $\mathcal{B}$ is the witness buffer. Relabeling recomputes every affected partial-trace target from the new semantic predicate, rather than inheriting an old label. Invalidation requires the forecaster to be retrained or recalibrated only on targets associated with the committed grammar version. A controller cannot mix a forecaster score from one predicate version with guards from another.

3.2 REPAIR, BLOCK, AND ESCALATE

At a partial trace $\tau_{\le t}$, the runtime compiler evaluates hard guards and a version-matched risk score $f^v(\tau_{\le t})$. If neither activates, the controller executes the proposed action. Otherwise it enumerates a fixed width set $\mathcal{R}_t$ of continuations from the safe prefix. A continuation is admissible only when every predicate in $C_{\mathrm{rep}}(\mathcal{H}^v)$ evaluates to zero and the task checker accepts its state transition. Admissible continuations are ordered lexicographically by task-checker progress, safety margin, typed-event edit distance, and serialized action identifier.

Let $r_t^\star$ be the first admissible continuation under this ordering and let $\mathrm{Esc}(\tau_{\le t})$ be one when an activated record carries an escalation flag. The runtime action is

$$a_{\mathrm{run}}(\tau_{\le t}) = \begin{cases} \mathrm{execute}, & \text{if no guard activates and } f^v \le \tau_{\mathrm{risk}}, \\ \mathrm{repair}(r_t^\star), & \text{if } r_t^\star \text{ exists}, \\ \mathrm{escalate}, & \text{if } r_t^\star \text{ does not exist and } \mathrm{Esc} = 1, \\ \mathrm{block}, & \text{otherwise.} \end{cases}$$

Thus every trigger has one complete resolution path. The threshold, continuation width, and task checker are frozen before a held-out assessment; the predicate version is included in each decision record.

3.3 TRAINING PROJECTIONS

The exploration projection rewards novel witnesses while balancing predicate families. The repair projection constructs safe continuations from the prefix before a completed witness. The prediction projection labels a partial trace positive when a version-matched completed witness occurs in its continuation. These consumers share the same record identity but remain operationally distinct: a discovered witness can change the reward distribution, a repaired trace can change the executor target, and a relabeled prefix can change the forecaster target. The compiler and version checks are what make these changes coherent.

4 MATCHED EVALUATION CONTRACT

The Safety IR is evaluated as a systems-maintenance hypothesis. The core comparison holds fixed the base model, training budget, witness-generation access, repair search, forecaster architecture, task

Algorithm 1 Versioned Safety-IR update and runtime action

Require: Committed grammar $\mathcal{H}^v$, revision Δ, trace prefix $\tau_{\leq t}$, proposed action u_t
1: $\mathcal{H}^{v+1} \leftarrow$ VALIDATEANDCOMMIT$(\mathcal{H}^v, \Delta)$
2: $(\rho_{\text{exp}}, \rho_{\text{rep}}, \rho_{\text{pred}}, \rho_{\text{run}}) \leftarrow \Gamma(\mathcal{H}^{v+1})$
3: Relabel affected buffer entries with ρ_{pred} and invalidate version-mismatched forecaster targets
4: Stamp every emitted output and decision record with version $v + 1$
5: Dispatch $(\tau_{\leq t}, u_t)$ to the version-matched runtime procedure

checker, calibration access, and maintenance cap. Mantle maintains one registry and compiler. The StageLocal control maintains four independently versioned surfaces, one for each projection, while receiving the same resources.

Before drift, both systems must pass an identical frozen functional test suite. The maintenance ledger records the initial active concrete-predicate count for each system, every added, deleted, or semantic-change edit, the stage to which each edit applies, elapsed engineering time, and the revision author. The edit percentage divides the total concrete edits by the target interface's initial active-predicate count. These records make a shared update and a four-surface update comparable without collapsing their maintenance costs.

4.1 BINARY ENDPOINTS AND PAIRED UNCERTAINTY

For method m, seed s, and held-out evaluation set $\mathcal{D}_s = \{z_1, \ldots, z_{n_s}\}$, let $y_{m,s,i}$ be the binary catastrophic label for trajectory z_i. The seed rate is

$$\widehat{R}_{m,s} = \frac{\sum_{i=1}^{n_s} y_{m,s,i}}{n_s}.$$

Every displayed rate must therefore be accompanied by its integer numerator, denominator, seed identity, and aggregation rule. Mission completion and safe task success use the same requirement: their numerators count the declared binary endpoint over an explicit set of evaluated trajectories. A balanced hazard-coverage suite records its template count by family and computes coverage from that declared denominator.

For a paired Mantle–StageLocal comparison, the unit is a task-template cluster, and both methods execute the same cluster under the same seed. Calibration selects $\theta_c^\star$ without accessing final test labels. Held-out uncertainty resamples the complete paired cluster, retaining all method outcomes and raters attached to that cluster. Each bootstrap replicate recomputes the paired risk difference from the resampled clusters; percentile endpoints form the confidence interval. This protocol identifies the effective independent unit and preserves within-template, within-seed, and within-rater dependence.

4.2 EXTERNAL AND DERIVATIVE SETTINGS

The derivative protocol begins from a frozen source benchmark manifest. It constructs a new interface manifest by recording every API binding change, semantic side-effect change, and task-checker adaptation. Source task IDs, derived task IDs, transformation IDs, checker versions, and split assignments remain linked in the manifest. A derivative task is admitted only when its typed action sequence, completion checker, and isolation boundary are executable under the derived interface. For R-Judge-style records (Yuan et al., 2024), conversion requires a serialized tool call, an observable state transition, a task objective, and an executable checker; records missing any field are excluded before sampling.

External evaluation uses a disjoint calibration partition and a final held-out partition. The calibration partition may support threshold and candidate selection, but it is never reused for final endpoints. Before final evaluation, the protocol freezes the native-defense candidate set, software version, checkpoint, configuration, and selection rule. The selected defense and Mantle are then evaluated on the same held-out task–attack pairs. This makes external comparisons reproducible and distinguishes adaptation access from test access.

5 Discussion

Mantle identifies a concrete maintenance boundary for tool-agent safety. Predicate meaning is carried by a versioned typed record rather than copied informally among discovery, repair, prediction, and enforcement. The resulting compiler provides a direct audit trail from a changed interface binding to the four consumers that must change with it.

The maintenance frontier complements this design with a decision rule. It requires a finite candidate set, a fixed resource budget, a completion floor, and a held-out paired endpoint. This framing makes shared semantic propagation a testable systems property: the relevant comparison is the complete maintenance operation under matched resources.

6 Conclusion

Mantle casts tool-agent safety maintenance as the management of one typed, versioned semantic object. Its Safety IR compiles stable predicate identities into exploration, repair, prediction, and runtime control, while its update rule keeps all four consumers synchronized after interface drift. The maintenance frontier and matched evaluation contract turn that architectural choice into an auditable comparison. Together they provide a precise basis for designing and assessing safety stacks whose semantics must evolve with the tools they control.

Ethics Statement

Mantle concerns controls for high-impact tool actions, including destructive writes, credential exposure, and irreversible commitments. The methodology requires explicit task-scoped authorization, trace-observable safety checks, and escalation for unrepaired high-impact actions. This manuscript reports no human-participant study, human judgments, or personal data.

Reproducibility Statement

Definitions 1–4, Sections 3.1–4.2, and Algorithm 1 specify the representation, update transaction, runtime transition rule, candidate selection, matching requirements, denominators, and paired uncertainty procedure.

AI Use Statement

Generative AI was used only for manuscript writing, editing, and presentation. The authors reviewed the edits and remain responsible for the final content.

References

David Dalrymple, Joar Skalse, Yoshua Bengio, Stuart Russell, Max Tegmark, Sanjit Seshia, Steve Omohundro, Christian Szegedy, Ben Goldhaber, Nora Ammann, Alessandro Abate, Joe Halpern, Clark Barrett, Ding Zhao, Tan Zhi-Xuan, Jeannette Wing, and Joshua Tenenbaum. Towards guaranteed safe AI: A framework for ensuring robust and reliable AI systems. *arXiv preprint arXiv:2405.06624*, 2024. URL https://arxiv.org/abs/2405.06624.

Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, Andy Jones, Sam Bowman, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Nelson Elhage, Sheer El-Showk, Stanislav Fort, Zac Hatfield-Dodds, Tom Henighan, Danny Hernandez, Tristan Hume, Josh Jacobson, Scott Johnston, Shauna Kravec, Catherine Olsson, Sam Ringer, Eli Tran-Johnson, Dario Amodei, Tom Brown, Nicholas Joseph, Sam McCandlish, Chris Olah, Jared Kaplan, and Jack Clark. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv preprint arXiv:2209.07858*, 2022. URL https://arxiv.org/abs/2209.07858.

Suyu Ge, Chunting Zhou, Rui Hou, Madian Khabsa, Yi-Chia Wang, Qifan Wang, Jiawei Han, and Yuning Mao. MART: Improving LLM safety with multi-round automatic red-teaming. *arXiv preprint arXiv:2311.07689*, 2023. URL https://arxiv.org/abs/2311.07689.

Jiaming Ji, Tianyi Qiu, Boyuan Chen, Borong Zhang, Hantao Lou, Kaile Wang, Yawen Duan, Zhonghao He, Jiayi Zhou, Zhaowei Zhang, Fanzhi Zeng, Kwan Yee Ng, Juntao Dai, Xuehai Pan, Aidan O'Gara, Yingshan Lei, Hua Xu, Brian Tse, Jie Fu, Stephen McAleer, Yaodong Yang, Yizhou Wang, Song-Chun Zhu, Yike Guo, and Wen Gao. AI alignment: A comprehensive survey. *arXiv preprint arXiv:2310.19852*, 2023. URL https://arxiv.org/abs/2310.19852v1.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/e9df36b21ff4ee211a8b71ee8b7e9f57-Abstract-Conference.html.

Tong Mu, Alec Helyar, Johannes Heidecke, Joshua Achiam, Andrea Vallone, Ian Kivlichan, Molly Lin, Alex Beutel, John Schulman, and Lilian Weng. Rule based rewards for language model safety. *arXiv preprint arXiv:2411.01111*, 2024. URL https://arxiv.org/abs/2411.01111.

Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 3419–3448, 2022. doi: 10.18653/v1/2022.emnlp-main.225. URL https://aclanthology.org/2022.emnlp-main.225/.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/28e50ee5b72e90b50e7196fde8ea260e-Abstract-Conference.html.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, Rui Wang, and Gongshen Liu. R-Judge: Benchmarking safety risk awareness for LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1467–1490, 2024. doi: 10.18653/v1/2024.findings-emnlp.79. URL https://aclanthology.org/2024.findings-emnlp.79/.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/4410c0711e9154a7a2d26f9b3816d1ef-Abstract-Conference.html.

A Complete Runtime Procedure

Algorithm 2 expands the controller transition in Section 3.2. It is deliberately fail-closed: a version mismatch, an unresolved binding, or an empty admissible-repair set cannot silently execute a triggered action.

Algorithm 2 Runtime decision from a version-matched Safety IR

Require: Trace prefix $\tau_{\leq t}$, proposed action u_t, committed grammar $\mathcal{H}^v$, compiled outputs $\Gamma(\mathcal{H}^v)$

Verify that the registry, compiled outputs, and forecaster share version v
if the version check fails **then**
  Block and record a version-mismatch event
else
  $g \leftarrow \textsc{HardGuard}(\tau_{\leq t}, u_t, \rho_{\mathrm{run}})$
  $q \leftarrow f^v(\tau_{\leq t})$
  if $g = 0$ and $q \leq \tau_{\mathrm{risk}}$ **then**
    Execute u_t
  else
    $\mathcal{R}_t \leftarrow \textsc{EnumerateSafeContinuations}(\tau_{\leq t}, \rho_{\mathrm{rep}})$
    if $\mathcal{R}_t \neq \emptyset$ **then**
      Execute the lexicographically first continuation in $\mathcal{R}_t$
    else if $\textsc{EscalationFlag}(\tau_{\leq t}, u_t, \rho_{\mathrm{run}}) = 1$ **then**
      Escalate
    else
      Block
    end if
  end if
end if

B Evaluation Ledger Schema

A maintenance record contains an immutable interface identifier, grammar version, concrete-predicate identifier, edit operation, old and new typed bindings, semantic-change reason, stage assignment, elapsed time, and validation result. The paired evaluation ledger additionally contains task-template identifier, seed, method, candidate identifier, calibration or held-out split label, task completion label, catastrophic label, rater or checker identifier, and the predicate version used at execution. The schema supports recomputation of every reported numerator, denominator, paired comparison, and candidate-selection decision.

C Derivative-Task Construction Checklist

For each derivative suite, the protocol freezes a source manifest before adaptation and records (i) the source task and interface identifiers, (ii) each surface or semantic interface transformation, (iii) the resulting typed operation schema, (iv) the task checker and isolation rule, (v) the calibration/held-out assignment, and (vi) the sampling rule. The matched control receives the same source manifest, calibration partition, candidate budget, and held-out task list. These fields make interface drift a reconstructible transformation rather than an informal change description.