
A Minimal Symbolic Control Contract for Small Language-Model Agents

Muhammad Saim

k230708@nu.edu.pk

Department of Computer Science, School of Computing,
National University of Computer and Emerging Sciences (FAST-NUCES),
Karachi, 75030, Pakistan

Abstract

We present a responsibility contract for language-model proposals against stale inventory observations: the adapter owns modeled facts, the generator proposes, and a symbolic controller admits and commits atomically. Under a sound closed adapter, accepted transfers preserve encoded invariants; intent correctness and physical safety remain outside this guarantee. A reproducible CPU fixture checks the implementation and exposes conservative refusals. This is a design and comparison agenda, not a new validation algorithm or a small-model performance result.

1. Position and Responsibility Contract

External critics, feedback, contracts and shielding are established (Kambhampati et al., 2024; Gundawar et al., 2024; Meyer, 1992; Alshiekh et al., 2017). Symbolic orchestration should publish its *knowledge and commit boundaries*: feasible proposals can become inadmissible before execution. Table 1 specifies a resource-independent controller; small-model performance is unmeasured.

Component	Obligation and boundary
World adapter	Expose typed IDs, consistent state version and missing observation fields; own authoritative modeled facts.
Generator	Interpret or request clarification; propose typed action P using observation O ; possess no mutation interface.
Atomic controller	Parse schema; within one lock check allowlist, IDs, current version and pre-conditions; construct/check candidate; commit state and audit, or return typed rejection.
Recovery caller	Reobserve after stale rejection; at most two proposals through the same controller; stop or clarify.

Table 1. Admission and mutation share one atomic boundary; rejections return to the bounded caller.

State S contains immutable IDs, unique placements, assigned destinations, bounded capacities and revision v . Observation O projects S with v and declared omissions. Proposal P carries request ID, revision, action and typed parcel/source/target references. Transfers require actual source, distinct target, compatible location kinds, capacity and assigned delivery destination. Acceptance changes one placement and increments v ; refusal changes nothing.

2. Strongest Defensible Property

Assume valid initial state, sound complete admission predicates, immutable IDs/capacities, fixed assignments, monotone revisions and exclusive mutation through one lock spanning checks and effects. Unique placement persists; occupancy decreases at source and increases within target capacity. Other placements stay fixed. The lock permits sequential reasoning about concurrent operations (Herlihy & Wing, 1990). Induction preserves invariants over accepted traces. Unlocked mutation after a revision check would not suffice. Crashes and distributed effects are excluded.

3. Failure Cases and Implementation Checks

Typed rejections distinguish malformed, unsupported, unknown-entity, stale and precondition failures from infrastructure errors and clarification. Supported, schema-valid proposals with known references are refused on revision mismatch before semantic checks. Request IDs provide correlation only.

A proposal at $v = 7$ is refused after a competing load commits $v = 8$; even a round-trip restoring placement invalidates the old revision. Loading a full vehicle is refused without mutation. Ambiguous “move the parcel” yields scripted clarification; valid locker movement changes one placement. These are implementation traces, not model behavior.

All 20 frozen fixtures (five per category) matched expected decisions and states: 5 accepted, 15 refused; zero infrastructure failures or accepted invariant violations. All 18 tests passed, covering interleavings, malformed-input isolation and bounded recovery. An independent oracle checked 67392 candidates over 234 legal states (two parcels, four locations, bounded capacities 0–2, current revision 7, proposal revisions 6–8): 996 accepted, zero mismatches or invariant violations. These bounded checks are not independent research observations or a general proof.

4. Limitations and Falsifiable Agenda

The adapter bears the hardest obligation: unmodeled facts cannot be validated. Projected observations do not solve belief-state planning (Kaelbling et al., 1998); affordance grounding (Ahn et al., 2022) and reasoning/action interaction (Yao et al., 2023) address broader capabilities. Geometry, travel, weight and hostile code are excluded. Valid transfers may misunderstand intent. No LLM was run; no superiority or physical-safety claim follows.

Global revisions reject useful transfers after unrelated changes; continuous updates can exhaust recovery. Invariant preservation supplies no progress guarantee. Postcondition verification and server idempotency address non-atomic external failures (Mansoor et al., 2026); our stronger atomic assumption does not replace those mechanisms.

Compare global revisions, sound read-set validation and split check/commit fixtures under identical invariants and proposal budgets. Report violations, useful stale refusals, observations, recovery exhaustion and goals separately. One accepted invariant violation falsifies preservation; a supported known mismatched-revision acceptance falsifies freshness; validator bypass or a third proposal falsifies bounded recovery. The executable contract exposes assumptions and refusal costs.

References

- Ahn, M., et al. (2022). Do as I can, not as I say: Grounding language in robotic affordances. arXiv:2204.01691v2. From <https://arxiv.org/abs/2204.01691v2>.
- Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., & Topcu, U. (2017). Safe reinforcement learning via shielding. arXiv:1708.08611v2. From <https://arxiv.org/abs/1708.08611v2>.
- Gundawar, A., Valmeekam, K., Verma, M., & Kambhampati, S. (2024). Robust planning with compound LLM architectures: An LLM-Modulo approach. arXiv:2411.14484v1. From <https://arxiv.org/abs/2411.14484v1>.
- Herlihy, M. P., & Wing, J. M. (1990). Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12, 463–492. From <https://cs.brown.edu/~mph/HerlihyW90/p463-herlihy.pdf>.
- Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 99–134. From <https://people.csail.mit.edu/lpk/papers/aij98-pomdp.pdf>.
- Kambhampati, S., Valmeekam, K., Guan, L., Stechly, K., Verma, M., Bhambri, S., Saldyt, L., & Murthy, A. (2024). LLMs can’t plan, but can help planning in LLM-Modulo frameworks. arXiv:2402.01817v2. From <https://arxiv.org/abs/2402.01817v2>.
- Mansoor, I. K., Phadke, A., & Rana, P. (2026). Verified tool calls improve LLM agent reliability under non-atomic failures. arXiv:2608.02645v1. From <https://arxiv.org/abs/2608.02645v1>.
- Meyer, B. (1992). Applying “design by contract”. *Computer*, 25, 40–51. From https://files.ifi.uzh.ch/rrerg/amadeus/teaching/courses/ase_fs10/Meyer1992.pdf.
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. arXiv:2210.03629v3. From <https://arxiv.org/abs/2210.03629v3>.