What Do We Really Need? Identifying Minimal Sufficient Temporal Properties via Refinement

Luca Brodo
luca.brodo@hshl.de
Hochschule Hamm-Lippstadt
Lippstadt, NRW, Germany

Giuseppe Scalora
giuseppe.scalora@hshl.de
Hochschule Hamm-Lippstadt
Lippstadt, NRW, Germany

Stefan Henkler
stefan.henkler@hshl.de
Hochschule Hamm-Lippstadt
Lippstadt, NRW, Germany

ABSTRACT

Formal verification relies on specifications formalized in CTL to express desired system behaviors. In practice, specification suites grow and evolve during iterative design cycles, accumulating overlapping and semantically dependent properties that increase engineering and verification cost. We propose to address this *specification debt* via *property refinement*, which captures semantic relationships between properties, identifying when one property is already guaranteed by another. This provides engineering insights into these semantic relationships, easing requirements management and the identification of redundant properties. Our evaluation shows reductions of up to 70% and speedups of up to 30 times, showcasing the practical potential in optimising verification workflows and guiding engineers towards identifying what is really needed.

CCS CONCEPTS

• **Software and its engineering → Model checking**.

KEYWORDS

Formal Verification, Property Refinement, CTL, Model Checking

ACM Reference Format:
Luca Brodo, Giuseppe Scalora, and Stefan Henkler. 2026. What Do We Really Need? Identifying Minimal Sufficient Temporal Properties via Refinement . In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-Companion '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3774748.3795661

1 INTRODUCTION

Formal verification is widely used to ensure that system models satisfy behavioral specifications expressed as temporal logic properties. In modern development workflows, specification suites grow and evolve as systems change through iterative design cycles, architectural changes, and collaboration across teams. Specifications are frequently extended, revised, or inherited from legacy designs, leading to suites with overlapping and redundant properties. The current practice of treating specifications as flat, independent lists, where each property is verified in isolation, ignores any semantic relationships and leads to the verification of properties whose satisfaction is already guaranteed by others. Beyond this unnecessary verification cost, this approach obscures the structure of system requirements and increases maintenance effort, as engineers focus on redundant properties, effectively accumulating *specification debt*.

We address this problem by introducing a formal notion of *property refinement* for properties expressed in Computational Tree Logic (CTL), a widely used logic due to its expressivess and efficient model checking algorithms [2]. A property φ' refines φ if it is strictly stronger: every behavior satisfying φ' also satisfies φ, hereby guaranteeing that verifying φ' suffices to ensure φ.

The closest related work is property subsumption as of [1], which identifies implication relations among temporal logic properties. However, subsumption is typically defined purely as logical implication, which is unsafe for omitting properties: unsatisfiable properties trivially imply all others while providing no correctness guarantees. Existing approaches therefore rely on model checking to validate implications, making them expensive, model-dependent, and unsuitable for early design stages. In contrast, our refinement notion is defined via trace inclusion with a non-emptiness constraint, ensuring correctness when refined properties replace weaker ones. This yields a model-agnostic analysis that can be applied independently of system models and reused across design iterations. Moreover, refinement induces a semantic structure over specification suites, exposing hierarchy and dependency relations that enable scalable redundancy identification as well as provide actionable insights into the evolution and health of system requirements.

2 REFINEMENT ANALYSIS

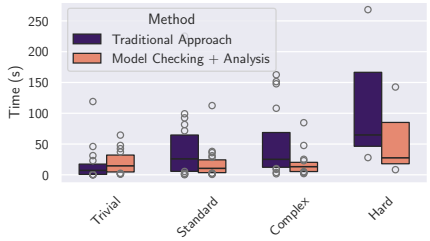
We define refinement between CTL properties as follows:

Definition 2.1 (Property Refinement). Let φ' and φ be CTL properties and $[\![\cdot]\!]$ denote the set of satisfying execution traces. We say that φ' *refines* φ, written as $\varphi' \preceq^{+} \varphi$, iff $[\![\varphi']\!] \subseteq [\![\varphi]\!]$ and $[\![\varphi']\!] \neq \emptyset$.

This definition reflects the understanding that φ' is more refined whenever every execution satisfying φ' also satisfies φ. The non-emptiness condition ensures that refinement is meaningful, excluding trivially false properties, which would otherwise refine every property via the empty set of traces. The non-emptiness guard is crucial in practice to make sure abstract properties can be substituted by refined ones without loss of correctness. The refinement relation can be proved to form a partial order, i.e. it is reflexive, transitive, and antisymmetric. Antisymmetry ensures that if two properties refine each other, they are logically equivalent. Transitivity allows intermediate properties in refinement chains to be eliminated, reducing analysis effort. They also provide a principled way to handle the cases of properties mutually refining each other: through transitivity and antisymmetry we can conclude that all properties are equivalent, allowing the set be

Please use nonacm option or ACM Engage class to enable CC licenses
This work is licensed under a Creative Commons Attribution 4.0 International License.
ICSE-Companion '26, April 12–18, 2026, Rio de Janeiro, Brazil
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2296-7/2026/04
https://doi.org/10.1145/3774748.3795661

Figure 1: Verification time comparison



collapsed into a single representative.Based on refinement, we propose the following procedure to extract a *minimal sufficient subset (MSS)* from a specification suite, i.e. the smallest subset of properties whose satisfaction guarantees the one of the original suite. First, properties are partitioned into classes based on shared atomic propositions, since properties that do not share propositions cannot refine each other. For each class, we analyze pairs of properties (ψ, ϕ) to determine whether $\psi \preceq^{+} \phi$ holds by constructing the corresponding alternating tree automaton and checking for language inclusion. This would involve checking for emptiness of the automaton corresponding to $\phi \wedge \neg\psi$, which is EXPTIME-complete [2] as it is equivalent to checking for satisfiability, so we use simulation relation between automata as a sound and polynomial-time approximation [4]. Whenever $\psi \preceq^{+} \phi$ holds, we expand the refinement set of ψ with that of ϕ to reduce future checks through transitivity. Finally, we identify the MSS by selecting properties that are not refined by any other.

3 PRELIMINARY EVALUATION

To assess the practical impact of the refinement analysis, we conducted our evaluation along two dimensions: (i) property reduction, and (ii) end-to-end verification savings.

Table 1 summarizes the reduction achieved across 69 benchmark suites we extracted from the MCC and the RERS 2019 Challenge [3, 5]. Our approach reduces the number of properties by an average of 36.6%, with reductions of up to 70%. On MCC benchmarks, which contain large sets of auto-generated CTL properties, mimicking a scenario where different tools or defensive programming practices are used to specify requirements during design iterations, we achieve an average reduction of 42.3%, confirming the prevalence of redundancy in such settings. Similarly, the RERS benchmarks, show an average reduction of 42.2% in the standatd suites and 25.1% in the industrial variants, which contains a smaller number of more curated properties. These results highlight the method's potential in identifying and eliminating redundant specifications even for the more carefully constructed suites. These relationships also provide valuable engineering insights, a form of *qualitative benefit*, as engineers will be able to trace them back at the requirement level identifying where and how redundancy and dependencies arise.

To evaluate end-to-end performance, we compared traditional model checking against a combination of refinement analysis with model checking on the MSS. To this end, we used synthetic benchmarks where we systematically varied *model complexity*, measured in the size of the corresponding transition system, and *property complexity*, referring to number of refinements, property length and temporal operator nesting. Figure 1 reports our results. For trivial models and suites with very few refinements, the traditional approach is slightly faster, as the analysis overhead is not offset by verification savings. However, as complexity increases, our approach consistently outperforms the traditional one. For complex and hard models, we observe substantial improvements, with verification time reduced by 10–30%, with peaks up to 30×.

Table 1: Summary of specification reduction results.

Category	# Suites	Avg. Red.	Avg. Time	Time Per Property	Highest Red.
RERS	9	42.2%	0.2s	0.01s	60%
RERS (Ind.)	30	25.1%	3.4s	0.17s	70%
MCC	30	42.3%	510.8s	1.06s	50%
Overall	**69**	**36.6%**	**171.0s**	**0.4s**	**60%**

Overall, the results show that refinement-based reduction has the potential to bring immediate benefits in terms of reduction and delivers significant end-to-end performance gains for non-trivial verification tasks. Moreover, even when models are still at the infant stage, the identification of a minimal sufficient property set exposes hidden dependencies among requirements, directly addressing specification debt and improving maintainability.

4 OUTLOOK

Several extensions of this work remain open. Firstly, a comparison with exact refinement checking involving satisfiability checking would be valuable to quantify trade-offs between precision and performance. Additionally, syntactic heuristics could be introduced to reduce overhead. Extending refinement-based analysis to richer logics such as timed CTL is another promising direction. Finally, integrating refinement analysis into existing verification toolchains to expose refinements at the requirements level would facilitate practical adoption and further evaluation. Addressing these directions, together with gathering feedback and exploring additional avenues, promises to fully realize the potential of refinement-based techniques in practical verification workflows.

ACKNOWLEDGMENTS

Supported by the Ministry of Economic Affairs, Industry, Climate Action and Energy of the State of North Rhine-Westphalia. Co-funded by the European Union. Grant number EFRE-20800498.

REFERENCES

[1] Rohit Dureja and Kristin Yvonne Rozier. 2018. More Scalable LTL Model Checking via Discovering Design-Space Dependencies (D^3). In *Tools and Algorithms for the Construction and Analysis of Systems*, Dirk Beyer and Marieke Huisman (Eds.). Springer International Publishing, Cham, 309–327.

[2] A. Prasad Sistla E. Allen Emerson. 1984. Deciding Branching Time Logic. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 175–184.

[3] Falk Howar, Marc Jasper, Malte Mues, David Schmidt, and Bernhard Steffen. 2021. The RERS challenge: towards controllable and scalable benchmark synthesis. *Int. J. Softw. Tools Technol. Transf.* 23, 6 (Dec. 2021), 917–930.

[4] Marcin Jurdziński. 2000. Small Progress Measures for Solving Parity Games. In *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*. 290–301.

[5] F. Kordon, P. Bouvier, H. Garavel, L. M. Hillah, F. Hulin-Hubard, N. Amat., E. Amparore, B. Berthomieu, S. Biswal, D. Donatelli, F. Galla, , S. Dal Zilio, P. G. Jensen, C. He, D. Le Botlan, S. Li, , J. Srba, Y. Thierry-Mieg, A. Walner, and K. Wolf. 2021. Complete Results for the 2021 Edition of the Model Checking Contest.