A Formal Semantic of Property Refinement in Temporal Logic for Formal Verification

Name Last name
University
University
{name.last_name}@university.com
City, Country

***Abstract*—Model checking against temporal logic specifications is a cornerstone of formal verification. While the refinement of models is a well-studied area, a formal groundwork for refinement at the specification level has been largely missing. This paper addresses this gap by introducing a rigorous, semantic notion of *property refinement* and establishing the fundamental algebraic properties of this refinement relation (e.g., as a pre-order). We leverage these properties to develop a methodology for optimizing verification. Our approach identifies a minimal, most-refined subset of specifications whose satisfaction refines that of the entire original suite. This is achieved by first partitioning specifications into equivalence classes based on shared atomic propositions, followed by an analysis of inter-class refinement links to construct the minimal subset. Empirical evaluation using specifications derived from real-world systems confirms our approach can reduce the number of formulas needing explicit verification by up to 89.45%, while incurring little-to-no computational overhead. This work promises more efficient model checking workflows without compromising verification rigor.**

***Index Terms*—Formal Methods, Model Checking, Linear Temporal Logic, Computation Tree Logic, Specification, Refinement**

I. Introduction

Model checking has become a cornerstone of formal verification for complex software systems, providing automated techniques to ensure their correctness against formal specifications. These specifications are typically expressed in temporal logics like Linear Temporal Logic (LTL) and Computation Tree Logic (CTL), which are adept at capturing the dynamic behavior, concurrency aspects, and interaction protocols inherent in modern software systems. While the field has seen extensive research in model refinement([1], [2], [3]) –allowing engineers to rigorously transition from abstract to concrete implementations while preserving critical properties– the analogous concept of property refinement at the specification level has remained significantly underdeveloped. Current practice involves checking a system against a large suite of properties, often derived independently from diverse requirements documents. These properties, however, are rarely isolated. Instead, they frequently exhibit intricate logical, structural, or semantic relationships reflecting the interconnectedness of software components and functionalities. Structurally, a high-level property concerning an entire software module might be decomposed into, and thus refined by, several properties governing its constituent sub-modules or internal state transitions. Consider a complex software protocol where a general liveness property (e.g., "a request will eventually be processed") could be refined by more specific properties detailing the successful completion of intermediate protocol stages. Consequently, the satisfaction of a carefully selected, stronger subset of refined properties would be sufficient to rigorously demonstrate the system's overall correctness against its comprehensive specification. However, a formal, semantically grounded framework for understanding and leveraging these potential refinement relationships between temporal logic properties themselves is largely missing. While Pnueli's work touched upon specification refinement [4], a comprehensive theory about refinement of specifications, particularly one that can be operationalized to optimize verification, has not emerged. Existing approaches often treat specification relationships informally, rely on purely syntactic strengthening without a deep semantic basis, or focus on property strengthening for specific applications without a general theory of property refinement.

This paper addresses this critical gap by introducing a formal, semantic definition of *property refinement* for LTL and CTL properties. We define what it means for one property to refine another based on traces and states inclusion. By establishing the algebraic properties of this refinement relation (e.g., reflexivity, transitivity, composability), we prove that this refinement relation forms a compositional preorder, enabling modular reasoning about specifications' strength. This is then leveraged to develop a technique to identify a minimal, most-refined subset of properties whose verification implies the satisfaction of the entire original specification set. This promises to significantly alleviate the model checking burden by eliminating redundant verifications, leading to more efficient verification workflows without sacrificing rigor. Our evaluation using specifications derived from real-world systems confirms our approach can reduce the number of formulas needing explicit verification by up to 89.45%, while incurring little-to-no computational overhead.

Our work, therefore, aims to elevate property refinement to the same level of formal understanding as model refinement and leverage it for efficient verification, compositional reasoning, and better management of complex specifications suites.

The remainder of this paper is structured as follows. Section II reviews existing literature on model and specification

refinement, highlighting the gap our work addresses. Section III provides essential background on LTL, CTL, and transition systems. In Section IV, we formally define LTL property refinement, establish its key structural properties such as reflexivity, transitivity, anti-symmetry, and composability, and discuss its automata-theoretic characterization. Section V extends this notion to CTL. Section VI details our approach for leveraging property refinement to achieve more efficient model checking. Section VII presents an empirical evaluation of our approach using real-world case studies, demonstrating its effectiveness in reducing the number of properties requiring verification with modest computation overhead. Finally, Section VIII concludes the paper and outlines potential avenues for future research.

II. Related Work

The concept of refinement in system modeling has been extensively studied, particularly through automata-theoretic notions such as bisimulation and simulation [5], [6], [7], [8], [9]. These notions provide formal frameworks to compare system behaviors at varying levels of abstraction, allowing for rigorous definitions of when one model can be considered a correct implementation or refinement of another. Bisimulation, in particular, characterizes behavioral equivalence by ensuring that two systems can simulate each other's transitions step-by-step, while simulation defines a preorder relation reflecting implementation correctness. Such refinement relations have become fundamental tools in the verification and design of reactive and concurrent systems, as they enable modular reasoning and compositional verification. Despite the maturity of refinement concepts for models, the extension of these ideas to system specifications—especially those expressed in temporal logics like LTL and CTL—is less developed. While the concept of specification refinement is present in some key work by Pnueli [4], it lacks comprehensive formalization or focuses on its properties, indicating an explored ground for further research. Moreover, other contributions address the notion of refinement [10], [11], but do so without explicitly framing it within the context of temporal logic. These gaps highlight a need for a more rigorous formalization of specification refinement within these temporal frameworks.

A related concept is that of property strengthening, which implicitly leverages property refinement techniques. For example, the approach proposed in [12] utilizes property strengthening to ensure that system properties are stricter, thereby supporting more robust verification guarantees. Nonetheless, these works typically stop short of providing a formal and comprehensive definition of property refinement itself, and they do not fully characterize the theoretical properties that would enable its practical exploitation. Compared to the related work, our approach departs from the traditional focus on model-level refinement by introducing a formal and compositional notion of refinement at the specification level, specifically for properties expressed in LTL and CTL . While existing literature extensively addresses behavioral equivalence and simulation-based refinement for transition systems, there is a noticeable lack of formal treatment regarding temporal property refinement and its characteristics. Prior works either treat specification refinement informally or rely on syntactic strengthening without providing a semantic foundation. In contrast, we define property refinement in terms of trace and state inclusion, develop its algebraic structure, and establish its compositionality. Furthermore, we leverage this notion to optimize model checking through refinement-based reduction of verification obligations, thus enabling more efficient but still rigorous verification workflows

III. Preliminaries

Linear Temporal Logic (LTL) was first introduced by Pnueli in his seminal work [13], where he proposed temporal logic as a formalism for specifying properties of concurrent systems. Since then, LTL has become a cornerstone in the field of formal methods, particularly in model checking and formal verification. We provide below its semantic over words.

Definition 1. LTL semantics over words

Let φ be an LTL formula over the set of atomic propositions AP. The linear-time property induced by φ is defined as:

$$\text{Words}(\varphi) = \left\{\sigma \in (2^{AP})^\omega \mid \sigma \models \varphi\right\}$$

where the satisfaction relation $\models \subseteq (2^{AP})^\omega \times \text{LTL}$ is the smallest relation satisfying the following rules:

- $\sigma \models \text{true}$
- $\sigma \models a \iff a \in A_0$ (i.e., $A_0 \models a$)
- $\sigma \models \varphi_1 \wedge \varphi_2 \iff \sigma \models \varphi_1$ and $\sigma \models \varphi_2$
- $\sigma \models \neg\varphi \iff \sigma \not\models \varphi$
- $\sigma \models \bigcirc\varphi \iff \sigma[1..] = A_1A_2A_3\ldots \models \varphi$
- $\sigma \models \varphi_1 \,\mathcal{U}\, \varphi_2 \iff \exists j \geq 0.\ \sigma[j..] \models \varphi_2$ and $\sigma[i..] \models \varphi_1$ for all $0 \leq i < j$

The concept of transition systems as the underlying semantic model for reactive systems is standard in formal verification. Transition systems are formally defined in foundational texts like [14] and [15], and are widely used to represent the behavior of systems under verification.

Definition 2. Transition system

A transition system $\mathcal{T}$ is a tuple

$$\mathcal{T} = (S, Act, \rightarrow, I, AP, L)$$

where:

- S is a (possibly infinite) set of states.
- Act is a set of actions.
- $\rightarrow \subseteq S \times Act \times S$ is the transition relation.
- $I \subseteq S$ is the set of initial states.
- AP is a set of atomic propositions.
- $L : S \rightarrow 2^{AP}$ is a labeling function that assigns to each state the set of atomic propositions true in that state.

The definition of satisfaction of an LTL formula by a transition system, $\mathcal{T} \models \varphi$, as trace inclusion $\text{Traces}(\mathcal{T}) \subseteq \text{Words}(\varphi)$, aligns with the semantics presented in these works.

Definition 3. A Transition system $\mathcal{T}$ satisfies an LTL property written as $\mathcal{T} \models \varphi$ is equivalent to saying:

$$\begin{aligned}\mathcal{T} \models \varphi &\iff \text{Traces}(\mathcal{T}) \subseteq \text{Words}(\varphi) \\ &\iff \mathcal{T} \models \text{Words}(\varphi) \\ &\iff \pi \models \varphi \quad \text{for all } \pi \in \text{Paths}(\mathcal{T}) \\ &\iff s_0 \models \varphi \quad \text{for all initial states } s_0\end{aligned}$$

Computation Tree Logic (CTL) was introduced by Clarke and Emerson in [16] as a branching-time temporal logic to specify properties of reactive systems. Unlike LTL, which quantifies over linear time (paths), CTL formulas express properties over branching structures of computations.

Definition 4. CTL semantics over transition systems

Let φ be a CTL formula over the set of atomic propositions AP, interpreted over a transition system $\mathcal{T} = (S, Act, \rightarrow, I, AP, L)$. The satisfaction relation $\models \subseteq S \times \text{CTL}$ is defined inductively as follows:

- $s \models \text{true}$ for all $s \in S$
- $s \models a \iff a \in L(s)$ for $a \in AP$
- $s \models \neg\varphi \iff s \not\models \varphi$
- $s \models \varphi_1 \wedge \varphi_2 \iff s \models \varphi_1$ and $s \models \varphi_2$
- $s \models \exists \bigcirc \varphi \iff \exists s'$ with $s \rightarrow s'$ and $s' \models \varphi$
- $s \models \forall \bigcirc \varphi \iff \forall s'$ with $s \rightarrow s', s' \models \varphi$
- $s \models \exists[\varphi_1 \mathcal{U} \varphi_2] \iff \exists \pi = s_0 s_1 \ldots$ starting at $s_0 = s$ such that $\exists j \geq 0$ with $s_j \models \varphi_2$ and $\forall 0 \leq i < j, s_i \models \varphi_1$
- $s \models \forall[\varphi_1 \mathcal{U} \varphi_2] \iff \forall \pi = s_0 s_1 \ldots$ starting at $s_0 = s, \exists j \geq 0$ with $s_j \models \varphi_2$ and $\forall 0 \leq i < j, s_i \models \varphi_1$

A *partially ordered set* (poset) $(L, \sqsubseteq)$ consists of a set L and a binary relation $\sqsubseteq$ that is reflexive, transitive, and anti-symmetric. Elements of L are said to be ordered under $\sqsubseteq$.

For a subset $B \subseteq L$, an element $y \in L$ is an *upper bound* of B if $s \sqsubseteq y$ for all $s \in B$. It is a *least upper bound* (lub) if $y \sqsubseteq y'$ for every upper bound y' of B. *Lower bounds* and the *greatest lower bound* (glb) are defined dually.

A *lattice* is a poset in which every non-empty finite subset has both a least upper bound and a greatest lower bound. A *complete lattice* is one in which every subset (finite or infinite) has both a lub and a glb. All finite lattices are complete.

Given a lattice $(L, \sqsubseteq)$ and a subset $B \subseteq L$, the (reflexive) *upward closure* of B is defined as:

$$\mathsf{UC}(B) = \{e \in L \mid \exists b \in B.\, b \sqsubseteq e\},$$

noting that $B \subseteq \mathsf{UC}(B)$.

IV. LTL Property Refinement and Its Structural Properties

In this section, we formalize a notion of refinement between LTL properties, which captures when one property strengthens or constrains another. This concept is foundational in compositional reasoning, system specification, and abstraction refinement. The refinement relation allows us to understand how a more concrete or detailed property (e.g., arising from an implementation or a strengthened specification) relates to a more abstract one (e.g., from an initial design or high-level requirement).

We define the refinement relation in terms of the language of infinite words that satisfy an LTL property, enabling a precise, model-theoretic interpretation. Following this, we establish fundamental structural properties of this refinement relation, such as logical implication equivalence, reflexivity, transitivity and anti-simmetry. These results lay the groundwork for reasoning about system behavior under refinement, as well as for modular verification where properties evolve or are composed incrementally. We formally define the refinement relation between LTL properties.

Definition 5 (LTL Property Refinement)**.** Let φ' and φ be two LTL properties. We say φ' refines φ, denoted $\varphi' \preceq \varphi$, if

$$\text{Words}(\varphi') = \{\sigma \in \text{Words}(\varphi) \mid \sigma \models \varphi'\}.$$

This definition immediately implies that refinement corresponds to language inclusion and logical implication.

Claim 1. The relation $\varphi' \preceq \varphi$ implies $\text{Words}(\varphi') \subseteq \text{Words}(\varphi)$.

Proof. Follows directly from Definition 5, as the right-hand side defines a subset of $\text{Words}(\varphi)$. ■

Claim 2. $\varphi' \preceq \varphi \iff \varphi' \implies \varphi$.

Proof. ($\Rightarrow$) Assume $\varphi' \preceq \varphi$. By Definition 5, any $\sigma \in \text{Words}(\varphi')$ must satisfy both $\sigma \models \varphi'$ and $\sigma \models \varphi$. Thus, if $\sigma \models \varphi'$, then $\sigma \models \varphi$, which means $\varphi' \implies \varphi$.

($\Leftarrow$) Assume $\varphi' \implies \varphi$. Then, any word σ satisfying φ' also satisfies φ, so $\text{Words}(\varphi') \subseteq \text{Words}(\varphi)$. The set $\{\sigma \in \text{Words}(\varphi) \mid \sigma \models \varphi'\}$ is therefore precisely $\text{Words}(\varphi')$. By Definition 5, $\varphi' \preceq \varphi$. ■

We now analyze the algebraic structure induced by the refinement relation $\preceq$ over LTL properties. These structural insights are foundational for reasoning about specification hierarchies, property transformations, and correctness preservation.

Claim 3 (Algebraic Properties of $\preceq$)**.** The refinement relation $\preceq$ is a pre-order (reflexive and transitive) and is anti-symmetric modulo logical equivalence:

- **Reflexive:** For any LTL property φ, $\varphi \preceq \varphi$.
- **Transitive:** If $\varphi_1 \preceq \varphi_2$ and $\varphi_2 \preceq \varphi_3$, then $\varphi_1 \preceq \varphi_3$.
- **Anti-symmetric (modulo $\equiv$):** $\varphi_1 \preceq \varphi_2$ and $\varphi_2 \preceq \varphi_1 \iff \varphi_1 \equiv \varphi_2$.

Proof. **Reflexive:** For any φ, $\text{Words}(\varphi) = \{\sigma \in \text{Words}(\varphi) \mid \sigma \models \varphi\}$ holds trivially, satisfying Definition 5. **Transitive:** Assume $\varphi_1 \preceq \varphi_2$ and $\varphi_2 \preceq \varphi_3$. By Claim 1, $\text{Words}(\varphi_1) \subseteq \text{Words}(\varphi_2)$ and $\text{Words}(\varphi_2) \subseteq \text{Words}(\varphi_3)$, implying $\text{Words}(\varphi_1) \subseteq \text{Words}(\varphi_3)$. Since $\varphi_1 \preceq \varphi_2$, every $\sigma \in \text{Words}(\varphi_1)$ satisfies $\sigma \models \varphi_1$. Thus, $\text{Words}(\varphi_1) = \{\sigma \in \text{Words}(\varphi_3) \mid \sigma \models \varphi_1\}$, so $\varphi_1 \preceq \varphi_3$ by Definition 5. **Anti-symmetric:** If $\varphi_1 \preceq \varphi_2$ and $\varphi_2 \preceq \varphi_1$, then by Claim 1, $\text{Words}(\varphi_1) \subseteq \text{Words}(\varphi_2)$ and $\text{Words}(\varphi_2) \subseteq \text{Words}(\varphi_1)$. This

implies $\text{Words}(\varphi_1) = \text{Words}(\varphi_2)$, meaning $\varphi_1 \equiv \varphi_2$. Conversely, if $\varphi_1 \equiv \varphi_2$, then $\text{Words}(\varphi_1) = \text{Words}(\varphi_2)$, and both refinement directions hold by Definition 5 and Claim 2. ■

This result establishes that (LTL/ $\equiv, \preceq$) forms a partially ordered set (poset). This structure enables organizing specifications from abstract (weaker) to concrete (stronger), facilitating stepwise refinement and precise management of specification evolution.

A critical aspect of refinement is its *composability* with logical connectives, essential for modular reasoning in system design where specifications are often constructed compositionally.

Claim 4 (Composability with Conjunction)**.** If $\varphi_1 \preceq \varphi_2$, then for any LTL property φ_3, $\varphi_1 \wedge \varphi_3 \preceq \varphi_2 \wedge \varphi_3$.

Proof. By assumption, $\varphi_1 \preceq \varphi_2$, so by Definition 5,

$$\text{Words}(\varphi_1) = \{\sigma \in \text{Words}(\varphi_2) \mid \sigma \models \varphi_1\}.$$

Now consider:

$$\begin{aligned}\text{Words}(\varphi_1 \wedge \varphi_3) &= \{\sigma \in (2^{AP})^\omega \mid \sigma \models \varphi_1 \wedge \sigma \models \varphi_3\} \\ &= \text{Words}(\varphi_1) \cap \text{Words}(\varphi_3).\end{aligned}$$

Similarly,

$$\text{Words}(\varphi_2 \wedge \varphi_3) = \text{Words}(\varphi_2) \cap \text{Words}(\varphi_3).$$

Using the refinement assumption again:

$$\text{Words}(\varphi_1) = \{\sigma \in \text{Words}(\varphi_2) \mid \sigma \models \varphi_1\}.$$

Intersecting both sides with $\text{Words}(\varphi_3)$ gives:

$$\begin{aligned}&\text{Words}(\varphi_1) \cap \text{Words}(\varphi_3) \\ &= \{\sigma \in \text{Words}(\varphi_2) \mid \sigma \models \varphi_1\} \cap \text{Words}(\varphi_3) \\ &= \{\sigma \in \text{Words}(\varphi_2) \cap \text{Words}(\varphi_3) \mid \sigma \models \varphi_1\} \\ &= \{\sigma \in \text{Words}(\varphi_2 \wedge \varphi_3) \mid \sigma \models \varphi_1\}.\end{aligned}$$

Thus,

$$\text{Words}(\varphi_1 \wedge \varphi_3) = \{\sigma \in \text{Words}(\varphi_2 \wedge \varphi_3) \mid \sigma \models \varphi_1 \wedge \varphi_3\},$$

which matches the form of Definition 5, showing:

$$\varphi_1 \wedge \varphi_3 \preceq \varphi_2 \wedge \varphi_3.$$

■

Claim 5 (Composability with Disjunction 1)**.** If $\varphi_1 \preceq \varphi_2$, then for any LTL property φ_3, $\varphi_1 \vee \varphi_3 \preceq \varphi_2 \vee \varphi_3$.

Proof. By assumption, $\varphi_1 \preceq \varphi_2$, so by Definition 5,

$$\text{Words}(\varphi_1) \subseteq \text{Words}(\varphi_2).$$

Now consider:

$$\begin{aligned}\text{Words}(\varphi_1 \vee \varphi_3) &= \{\sigma \in (2^{AP})^\omega \mid \sigma \models \varphi_1 \vee \sigma \models \varphi_3\} \\ &= \text{Words}(\varphi_1) \cup \text{Words}(\varphi_3).\end{aligned}$$

Similarly,

$$\text{Words}(\varphi_2 \vee \varphi_3) = \text{Words}(\varphi_2) \cup \text{Words}(\varphi_3).$$

Using the refinement assumption again:

$$\text{Words}(\varphi_1) \subseteq \text{Words}(\varphi_2).$$

Hence,

$$\text{Words}(\varphi_1) \cup \text{Words}(\varphi_3) \subseteq \text{Words}(\varphi_2) \cup \text{Words}(\varphi_3),$$

which means

$$\text{Words}(\varphi_1 \vee \varphi_3) \subseteq \text{Words}(\varphi_2 \vee \varphi_3).$$

Therefore, by definition $\varphi_1 \vee \varphi_3 \preceq \varphi_2 \vee \varphi_3$. ■

Claim 6 (Composability with Disjunction 2)**.** If $\varphi_1 \preceq \varphi_2$ or $\varphi_1 \preceq \varphi_3$, then $\varphi_1 \preceq \varphi_2 \vee \varphi_3$.

Proof. We are given that either $\varphi_1 \preceq \varphi_2$ or $\varphi_1 \preceq \varphi_3$. We prove the claim in both cases.

- **Case 1:** Assume $\varphi_1 \preceq \varphi_2$. By Definition 5, we have:

  $$\text{Words}(\varphi_1) = \{\sigma \in \text{Words}(\varphi_2) \mid \sigma \models \varphi_1\}.$$

  Since $\text{Words}(\varphi_2) \subseteq \text{Words}(\varphi_2 \vee \varphi_3)$, it follows that

  $$\text{Words}(\varphi_1) \subseteq \text{Words}(\varphi_2 \vee \varphi_3).$$

  Moreover, for all $\sigma \in \text{Words}(\varphi_1)$, we have $\sigma \models \varphi_1$ by definition. Thus:

  $$\text{Words}(\varphi_1) = \{\sigma \in \text{Words}(\varphi_2 \vee \varphi_3) \mid \sigma \models \varphi_1\},$$

  which by Definition 5 implies $\varphi_1 \preceq \varphi_2 \vee \varphi_3$.
- **Case 2:** The proof is symmetric if $\varphi_1 \preceq \varphi_3$, since $\text{Words}(\varphi_3) \subseteq \text{Words}(\varphi_2 \vee \varphi_3)$, and the same reasoning applies.

In either case, we conclude that $\varphi_1 \preceq \varphi_2 \vee \varphi_3$. ■

Claim 7 (Refinement into Conjunction)**.** If $\varphi_1 \preceq \varphi_2$ and $\varphi_1 \preceq \varphi_4$, then $\varphi_1 \preceq \varphi_2 \wedge \varphi_4$.

Proof. Given $\varphi_1 \preceq \varphi_2$ and $\varphi_1 \preceq \varphi_4$, by Definition 5: $\text{Words}(\varphi_1) = \{\sigma \in \text{Words}(\varphi_2) \mid \sigma \models \varphi_1\}$ and $\text{Words}(\varphi_1) = \{\sigma \in \text{Words}(\varphi_4) \mid \sigma \models \varphi_1\}$.

Thus, if $\sigma \in \text{Words}(\varphi_1)$, then $\sigma \in \text{Words}(\varphi_2)$, $\sigma \in \text{Words}(\varphi_4)$, and $\sigma \models \varphi_1$. This means $\sigma \in \text{Words}(\varphi_2 \wedge \varphi_4)$ and $\sigma \models \varphi_1$, so $\text{Words}(\varphi_1) \subseteq \{\sigma \in \text{Words}(\varphi_2 \wedge \varphi_4) \mid \sigma \models \varphi_1\}$.

Conversely, if $\sigma \in \{\sigma \in \text{Words}(\varphi_2 \wedge \varphi_4) \mid \sigma \models \varphi_1\}$, then $\sigma \in \text{Words}(\varphi_2)$, $\sigma \in \text{Words}(\varphi_4)$, and $\sigma \models \varphi_1$. By the initial assumptions, this implies $\sigma \in \text{Words}(\varphi_1)$. Hence, $\{\sigma \in \text{Words}(\varphi_2 \wedge \varphi_4) \mid \sigma \models \varphi_1\} \subseteq \text{Words}(\varphi_1)$.

Therefore, $\text{Words}(\varphi_1) = \{\sigma \in \text{Words}(\varphi_2 \wedge \varphi_4) \mid \sigma \models \varphi_1\}$, meaning $\varphi_1 \preceq \varphi_2 \wedge \varphi_4$. ■

Finally, we establish the automata-theoretic characterization of LTL refinement, which is pivotal for its application in model checking. This correspondence allows leveraging standard automata-based techniques for verifying refinement.

Claim 8 (Automata-Theoretic Characterization of LTL Refinement)**.** Let φ' and φ be two LTL properties, with corresponding NBAs $A_{\varphi'}$ and A_φ such that $L(A_\psi) = \text{Words}(\psi)$. Then,

$$\varphi' \preceq \varphi \iff L(A_{\varphi'}) \subseteq L(A_\varphi).$$

Proof. By the correctness of the NBA construction for LTL formulas, the language accepted by A'_φ coincides exactly with the set of infinite words satisfying φ':

$$L(A_\varphi) = \text{Words}(\varphi) \text{ and } L(A_{\varphi'}) = \text{Words}(\varphi').$$

We prove both directions:

($\Rightarrow$) Assume $\varphi' \preceq \varphi$. By the definition of refinement for LTL properties, this means

$$\text{Words}(\varphi') \subseteq \text{Words}(\varphi).$$

Using the correspondence with their NBAs, we get

$$L(A'_\varphi) \subseteq L(A_\varphi),$$

which by definition of refinement for NBAs implies

$$A'_\varphi \sqsubseteq A_\varphi.$$

($\Leftarrow$) Conversely, assume $A'_\varphi \sqsubseteq A_\varphi$. By definition, this means

$$L(A'_\varphi) \subseteq L(A_\varphi).$$

Using the correspondence with LTL formulas, it follows that

$$\text{Words}(\varphi') \subseteq \text{Words}(\varphi),$$

hence

$$\varphi' \preceq \varphi.$$

■

Claim 9 (Implication for Model Checking)**.** If $\varphi' \preceq \varphi$ and a transition system $TS \models \varphi'$, then $TS \models \varphi$.

Proof. $TS \models \varphi'$ implies $L(TS) \subseteq L(A_{\varphi'})$. Given $\varphi' \preceq \varphi$, by Proposition 8, $L(A_{\varphi'}) \subseteq L(A_\varphi)$. Therefore, $L(TS) \subseteq L(A_\varphi)$, which implies $TS \models \varphi$. ■

The LTL property refinement relation offers significant advantages for system design and verification. Firstly, it enables *substitutability*: a more refined (stronger) specification can replace a more abstract (weaker) one. This is valuable for evolving systems and in self-adaptive contexts, as components satisfying refined properties can safely substitute those meeting only the original, more abstract ones. Secondly, it facilitates *optimized verification*: by identifying a minimal subset of the most-refined formulas within a specification suite, the overall model checking effort can be substantially reduced by focusing only on this core stringent set, without compromising verification completeness.

V. CTL Property Refinement and Its Structural Properties

In this section, we extend the notion of refinement to Computation Tree Logic (CTL) properties, which, compared to LTL, operate over branching-time structures rather than linear traces. CTL refinement captures when one property strengthens or constrains another in the context of state-based reasoning.

A. Definition of CTL Refinement

We define CTL refinement in terms of state satisfaction over all possible transition systems, providing a model-theoretic interpretation that aligns with CTL's semantics.

Definition 6. Let φ' and φ be two CTL properties. We say φ' refines φ, written as $\varphi' \preceq \varphi$, if for every transition system $\mathcal{T}$ and every state $s \in S$,

$$\{s \in S \mid \mathcal{T}, s \models \varphi'\} \subseteq \{s \in S \mid \mathcal{T}, s \models \varphi\}.$$

This means that whenever a state satisfies φ' in any transition system, it must also satisfy φ.

Lemma 1. Let φ' and φ be two CTL properties. We say φ' refines φ in relation to a TTS $\mathcal{T}$, written as $\varphi' \preceq_\mathcal{T} \varphi$, if for the TTS $\mathcal{T}$ and every state $s \in S$,

$$\{s \in S \mid \mathcal{T}, s \models \varphi'\} \subseteq \{s \in S \mid \mathcal{T}, s \models \varphi\}.$$

B. Basic Properties of CTL Refinement

We now establish fundamental properties of the CTL refinement relation, analogous to those shown for LTL refinement.

Claim 10.

$\varphi' \preceq \varphi \Leftrightarrow \varphi' \implies \varphi$ is valid (true in all transition systems).

Proof. The proof follows directly from the definition:

($\Rightarrow$) If $\varphi' \preceq \varphi$, then for every $\mathcal{T}$ and state s, $\mathcal{T}, s \models \varphi'$ implies $\mathcal{T}, s \models \varphi$. Thus, $\varphi' \implies \varphi$ holds universally.

($\Leftarrow$) If $\varphi' \implies \varphi$ is valid, then whenever $\mathcal{T}, s \models \varphi'$, we must have $\mathcal{T}, s \models \varphi$, satisfying Definition 6.

■

Claim 11. The CTL refinement relation $\preceq$ is:

- **Reflexive:** For any CTL property φ, $\varphi \preceq \varphi$.
- **Transitive:** If $\varphi_1 \preceq \varphi_2$ and $\varphi_2 \preceq \varphi_3$, then $\varphi_1 \preceq \varphi_3$.
- **Anti-symmetric:** If $\varphi_1 \preceq \varphi_2$ and $\varphi_2 \preceq \varphi_1$, then $\varphi_1 \equiv \varphi_2$ (they are semantically equivalent).

Proof. The proof mirrors that of Claim 3, using state satisfaction instead of word languages:

- **Reflexivity:** Trivially, $\mathcal{T}, s \models \varphi$ implies $\mathcal{T}, s \models \varphi$.
- **Transitivity:** If $\varphi_1 \preceq \varphi_2$ and $\varphi_2 \preceq \varphi_3$, then $\mathcal{T}, s \models \varphi_1$ implies $\mathcal{T}, s \models \varphi_2$, which in turn implies $\mathcal{T}, s \models \varphi_3$.
- **Anti-symmetry:** If $\varphi_1 \preceq \varphi_2$ and $\varphi_2 \preceq \varphi_1$, then $\mathcal{T}, s \models \varphi_1$ iff $\mathcal{T}, s \models \varphi_2$ for all $\mathcal{T}$ and s, meaning they are equivalent.

■

C. Compositionality of CTL Refinement

Like in LTL, refinement in CTL is preserved under logical composition, which is crucial for modular reasoning.

Claim 12. Let $\varphi_1, \varphi_2, \varphi_3$ be CTL properties. If $\varphi_1 \preceq \varphi_2$, then:

$$\varphi_1 \wedge \varphi_3 \preceq \varphi_2 \wedge \varphi_3.$$

Proof. Assume $\mathcal{T}, s \models \varphi_1 \wedge \varphi_3$. Then:

- $\mathcal{T}, s \models \varphi_1$, which by refinement implies $\mathcal{T}, s \models \varphi_2$.

- $\mathcal{T}, s \models \varphi_3$.

Thus, $\mathcal{T}, s \models \varphi_2 \wedge \varphi_3$, proving the claim. ■

Claim 13. If $\varphi_1 \preceq \varphi_2$ or $\varphi_1 \preceq \varphi_3$, then:

$$\varphi_1 \preceq \varphi_2 \vee \varphi_3.$$

Proof. If $\varphi_1 \preceq \varphi_2$, then $\mathcal{T}, s \models \varphi_1$ implies $\mathcal{T}, s \models \varphi_2$, which further implies $\mathcal{T}, s \models \varphi_2 \vee \varphi_3$. The case for $\varphi_1 \preceq \varphi_3$ is symmetric. ■

D. Implications for Model Checking

CTL refinement ensures that satisfaction of a refined property implies satisfaction of the abstract property, which is key for hierarchical verification.

Claim 14. Let $\varphi' \preceq \varphi$ be CTL properties. For any transition system $\mathcal{T}$ and state s:

$$\mathcal{T}, s \models \varphi' \implies \mathcal{T}, s \models \varphi.$$

Proof. Direct from Definition 6. ■

This means that verifying φ' is sufficient to ensure φ holds, enabling efficient verification by focusing on refined specifications.

VI. Minimal Specification Subsets via Property Refinement

This section introduces our methodology for optimizing model checking by identifying a minimal subset of specifications whose verification implies the entire suite's correctness. Traditional independent verification of each formula is computationally burdensome for complex specifications; our approach, based on property refinement (Sections IV, V), mitigates this redundancy.

The key insight is that specifications often exhibit refinement relationships: satisfying a more refined (stronger) formula implies satisfaction of a less refined one. By identifying these, particularly among constituent subformulas, we focus verification on a minimal set of the strongest properties. This reduces model checking runs and leverages temporal logic's hierarchical nature.

Our two-stage methodology first groups specifications into equivalence classes by shared atomic propositions (a coarse filter). Subsequently, within each class, we analyze refinement relations among atomic subformulas, conceptualized as a lattice, to identify interdependencies.

The objective is a minimal subset of atomic subformulas that collectively refines all original formulas, guaranteeing full specification satisfaction with reduced workload. This is theoretically sound, drawing on established algebraic properties of refinement and lattice theory. The remainder of this section details the algorithms for constructing these equivalence classes (Algorithm 1), detecting refinement links between subformulas, and selecting the minimal subset of properties for verification (Algorithm 2). We also provide a comprehensive complexity analysis and discuss practical heuristics for ensuring scalability in complex verification scenarios.

A. Building the Classes

Algorithm 1 AP-based Equivalence Class Construction

Require: A set of LTL formulas $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ with corresponding atomic proposition sets $AP_1, \ldots, AP_n \subseteq \mathcal{A}$
Ensure: A set of equivalence classes $\mathcal{C}$

```
PropList ← extractAtomicPropositions(Φ)  ▷ PropList[i] = AP_i
Initialize a hash map L : A → List of indices
Initialize disjoint-set forest (Union-Find) UF with n elements
for i ← 1 to n do
    for all literal ℓ ∈ PropList[i] do
        L[ℓ] ← L[ℓ] ∪ {i}
    end for
end for
for all literal ℓ ∈ dom(L) do
    I ← L[ℓ]
    for j ← 2 to |I| do
        UF.Union(I[1], I[j])
    end for
end for
Initialize empty map C : Representative → List of Indices
for i ← 1 to n do
    r ← UF.Find(i)
    C[r] ← C[r] ∪ {i}
end for
return C
```

This algorithm partitions a given set of LTL formulas $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ into equivalence classes based on the atomic propositions they contain. Each formula φ_i is associated with a set of atomic propositions $AP_i \subseteq \mathcal{A}$. The algorithm first extracts the atomic propositions from all formulas and stores them in a list `PropList`, where `PropList`[i] corresponds to AP_i. It then initializes a hash map L that maps each atomic proposition $\ell \in \mathcal{A}$ to the list of formula indices that contain ℓ. A Union-Find data structure is employed to efficiently merge these indices into equivalence classes. For each formula, the algorithm updates L by associating its atomic propositions with its index. Next, for every atomic proposition ℓ, it retrieves the list of formulas that include ℓ and unions their indices in the Union-Find structure, indicating they belong to the same equivalence class. Finally, the algorithm groups formulas according to their Union-Find representatives and returns the resulting equivalence classes $\mathcal{C}$.This approach guarantees that formulas sharing at least one atomic proposition are clustered together, thereby pruning the search space by excluding refinement comparisons between formulas with disjoint atomic propositions.

This step is used solely to prune the search space for the subsequent refinement analysis and can be viewed as a relatively coarse filtering mechanism. The algorithm outputs a set of equivalence classes $\mathcal{C}$. If the number of classes equals the number of formulas ($|\mathcal{C}| = n$), it implies that none of the

properties can be a refinement of another. This follows from the fact that, for one property to refine another, they must share at least some common atomic propositions (literals). Moreover, refinement checks only need to be performed within classes that contain two or more specifications (i.e., those with $|\mathcal{C}_i| \geq 2$), as classes with a single element cannot contain refinement relations between different formulas.

B. Finding refinements

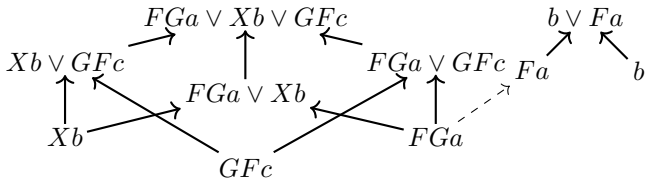

Figure 1: Example of subformula decomposition and refinement links. Dashed arrow indicates an inter-formula refinement.

To determine refinement relations between temporal logic formulas, we adopt a structured approach based on the decomposition of formulas into hierarchical levels of subformulas. Consider, for example, the diagram in Figure 1, where formulas are decomposed into increasingly finer subformulas. The key insight is that satisfaction of a lower-level subformula implies satisfaction of higher-level formulas through chains of refinement relations.

For instance, if a lower-level subformula such as FGa (eventually, always a) is a refinement of the subformula Fa (eventually a), then verifying FGa suffices to ensure the satisfaction of both the upper level formula of FGa itself and Fa. By identifying such refinement relations—represented, for example, by the dashed line in Figure 1—we can minimize the number of subformulas that need explicit model checking. Specifically, we prioritize the most refined subformulas at the lowest levels, as their satisfaction transitively guarantees the satisfaction of dependent formulas at higher levels.

Our objective is to systematically identify as many such refinement links as possible, thereby constructing a minimal subset of atomic subformulas whose verification ensures the satisfaction of all formulas in the specification. This approach optimizes the verification process by reducing redundancy and focusing on the most semantically influential subformulas.

To formalize the selection of the minimal subset, let $\varphi(\mathcal{C}) = \bigwedge_{\varphi \in \mathcal{C}} \varphi$ denote the conjunction of all LTL formulas in an equivalence class $\mathcal{C}$. For each formula $\varphi_i \in \mathcal{C}$, let S_i be the set of its subformulas. The refinement relation $\preceq$ (Definition 5) establishes a preorder on $\bigcup_i S_i$. The quotient $L = (\bigcup_i S_i)/\sim$, where $a \sim b \iff (a \preceq b \wedge b \preceq a)$, forms a poset. Given that L is finite and equipped with meet (conjunction) and join (disjunction), it constitutes a complete lattice (cf. Claim 3 and standard lattice theory results). Let $\Phi_i^{\perp}$ represent the set of most-refined elements (atomic subformulas) in the sub-lattice corresponding to formula φ_i.

Algorithm 2 Minimal Refined Subset Finder

Require: Set of atomic subformulas $\Phi^{\perp} = \Phi_1^{\perp}, \ldots, \Phi_n^{\perp}$ where each $\Phi_i^{\perp}$ is the set of most refined elements from formula i
Ensure: Minimal set $\phi^{\perp} \subseteq \bigcup_i \Phi_i^{\perp}$ that refines all formulas
$\phi^{\perp} \leftarrow \emptyset$
$covered \leftarrow \emptyset$ ▷ Tracks already refined formulas
$graph \leftarrow \emptyset$ ▷ Refinement relation graph
$sub \leftarrow \bigcup_{i=1}^{n} \Phi_i^{\perp}$ ▷ Collect all atomic subformulas
for $b \in sub$ **do**
  for $c \in sub \setminus b$ **do**
    if CHECKREFINEMENT(b, c) **then**
      $graph \leftarrow graph \cup (b, c)$
    end if
  end for
end for
Sort $\bigcup_i \Phi_i^{\perp}$ by descending out-degree in $graph$
for $b \in$ sorted $\bigcup_i \Phi_i^{\perp}$ **do**
  if $M \uparrow (b) \not\subseteq covered$ or $deg_{graph}(b) \neq 0$ **then**
    $\phi^{\perp} \leftarrow \phi^{\perp} \cup b$
    $covered \leftarrow covered \cup M \uparrow (b)$
    if $covered$ contains all formulas **then**
      return $\phi^{\perp}$
    end if
  end if
end for
return $\phi^{\perp}$

Algorithm 3 CheckRefinement Subroutine for LTL

Require: Two LTL formulas b, c
Ensure: Returns true if $b \preceq c$
Construct NBAs A_b and A_c for b and c respectively
if $L(A_b) \subseteq L(A_c)$ holds **then**
  return true
else
  return false
end if

Algorithm 4 CheckRefinement Subroutine for CTL

Require: Two CTL formulas φ', φ
Ensure: Returns true if $\varphi' \preceq \varphi$ (i.e., $\varphi' \implies \varphi$ is valid)
if $\varphi' \wedge \neg\varphi$ is unsatisfiable **then**
  return true
else
  return false
end if

Inter-lattice refinement links may exist between subformulas from different φ_i. Leveraging the transitivity of refinement (Claim 3) and its compositional properties (e.g., Claim 7), satisfaction of a lower-level atomic subformula b can imply the satisfaction of multiple higher-level formulas. We denote $M \uparrow (b) = \{\varphi \in \mathcal{C} \mid \exists \psi \in \text{sub}(\varphi) \text{ such that } b \preceq \psi\}$ as the set of formulas in $\mathcal{C}$ covered by the atomic subformula b. Our objective is to find a minimal set $\phi^{\perp} \subseteq \bigcup_i \Phi_i^{\perp}$ such that:

$$\bigwedge_{b \in \phi^{\perp}} \left(\bigwedge_{\varphi_k \in M \uparrow (b)} \varphi_k \right) \preceq \varphi(\mathcal{C}).$$

Essentially, the conjunction of properties covered by the elements in $\phi^{\perp}$ must refine the conjunction of all properties in the class. Algorithm 2 presents a constructive approach to find such a $\phi^{\perp}$. Once the relation graph is established, the algorithm sorts all candidate subformulas by their out-degree in descending order, prioritizing those that refine the greatest number of other subformulas. It then greedily and iteratively builds the minimal set by adding a subformula if its upperclosure has not yet been covered or if it actively refines other subformulas (has a non-zero out-degree), continuing this greedy selection until all original properties are accounted for by the chosen subset. We motivate the use of a greedy algorithm as it provides an efficient, asymptotically near-optimal approximation for the NP-Hard problem of minimal property set selection [17], [18], performing well with dense refinement graphs typical of redundant specifications.

C. Complexity of the algorithm

The computational complexity of Algorithm 2, excluding the CHECKREFINEMENT subroutine, is $O(m^2)$, where m is the maximum number of unique atomic subformulas across all $\varphi_i \in \mathcal{C}$, and $n = |\mathcal{C}|$. The $O(m^2)$ term arises from constructing the refinement graph (lines 5-11). The greedy selection process (lines 13-21, where $|M \uparrow (b)| \leq n$) was mainly chosen for its scalability. As a matter of fact, it has a complexity $O(mn)$, which is dominated by the previous term. The subroutine CHECKREFINEMENT, however, might dominate the complexity. For example, in the case of LTL formula, Verifying $b \preceq c$ typically involves constructing equivalent Non-deterministic Büchi Automata (NBAs), A_b and A_c, and checking for language inclusion $L(A_b) \subseteq L(A_c)$. While NBA complementation can be avoided by checking $L(A_b \cap A_{\neg c}) = \emptyset$, the construction of A_b (or $A_{\neg c}$) from an LTL formula is exponential in the formula length in the worst case, i.e., $O(2^{|\varphi|})$, where $|\varphi|$ is the length of the LTL formula. Consequently, the overall complexity of the refinement analysis is $O(m^2 \cdot C_{\text{check}})$, where $C_{\text{check}} = O(2^{\max(|\text{subformula}|)})$. If the atomic subformulas are consistently short, C_{check} approaches a small constant. In this regime, the $O(m^2)$ term, reflecting the number of pairwise comparisons, governs the scalability with respect to the number of unique subformulas.

This complexity can be mitigated through several heuristics. In addition to the filter provided by Algorithm 1, we suggest, for instance, to use syntactic refinement rules (e.g. $F\phi \sqsubseteq_{\text{syn}} F\psi$ if $\phi \sqsubseteq_{\text{syn}} \psi$),which provide a fast, sound, but incomplete method for establishing refinement, avoiding expensive NBA constructions. Furthermore, the explicit refinement graph (Algorithm 2, line 8) combined with the transitivity of $\preceq$ (Claim 3) allows to infer refinements, thereby reducing redundant CHECKREFINEMENT calls.

Considering this complexity, our refinement-based approach becomes particularly advantageous under the following conditions:

- **High Redundancy:** When many subformulas refine others (i.e., $m_{ref} \ll m_{abs}$, where m_{ref} is the size of $\phi^{\perp}$), the minimal set $\phi^{\perp}$ becomes significantly smaller than n, leading to fewer model checking runs.
- **Large Transition Systems:** For systems where $|TS|$ is large, the model checking cost $O(|TS| \cdot 2^{|\text{formula}|})$ for each formula is substantial. The upfront preprocessing cost of $O(m^2 \cdot 2^{\max(|\text{subformula}|)})$ can be amortized if it avoids multiple expensive model checking calls.
- **Reusable Subformulas:** The presence of frequently repeated subformulas across the specification suite leads to a denser refinement graph and greater opportunities for reuse of refinement checks.

In the context of CTL properties, the nature of the CHECKREFINEMENTCTL subroutine presents a critical choice.Determining general CTL refinement (i.e., validity of $\varphi' \preceq \varphi$ across all models) is an EXPTIME-complete problem in formula length. This potentially high cost for each pairwise check could render the overall refinement analysis prohibitive for CTL specifications if numerous, complex checks are required. An alternative is to perform *model-dependent* CTL refinement. In this scenario, given a specific transition system M, one checks if $\varphi' \preceq_M \varphi$ (c.f. Lemma 1) holds for all states *within that model* M. This involves one CTL model checking run on M as the validity of $\varphi' \preceq \varphi$ is equivalent to the unsatisfiability of $\varphi' \wedge \neg\varphi$, resulting in a much more tractable complexity of $O((|\varphi'| + |\varphi|) \cdot |M|)$. While this significantly reduces the cost of each individual refinement check, it introduces a caveat: the identified minimal set of refined CTL properties would only be guaranteed to imply the original specification suite *for that particular model* M (or its equivalence class found through bisimulation). If the underlying system model changes, the refinement relationships established model-dependently may no longer hold, potentially invalidating the correctness of the reduced property set for the new model. This contrasts with the LTL approach where refinement, based on language inclusion, is model-independent. Therefore, while model-dependent CTL refinement offers computational advantages for the analysis phase, its utility for deriving a universally applicable optimized set of specifications is limited to the context of the specific model used during the refinement analysis.

VII. EVALUATION

The increasing complexity of modern software systems, such as those envisioned for smart farming, presents significant verification challenges. These systems often require a large

Table I: Comparison between before and after optimization across various projects #F represents the number of properties.

System	Name	Original #F	New #F	Reduction (%)	Time (ms)
S1	High-Level description for system that collects, checks, and shares road weather data	512	54	89.45	1001.03
S2	Low-Level description for system that collects, checks, and shares road weather data	238	77	67.65	301.38
S3	Requirements for a unified university inventory system	404	51	87.38	1666.93
S4	Requirements for company specific inventory system	171	56	67.25	424.33
S5	Software for analyzing data for transportation applications	551	549	0.36	7.69
S6	Define software requirements for managing heating and cooling	183	174	4.92	14.19
S7	Tracking of water use and resource management	741	486	34.40	426.58
S8	Support secure exchange of tachograph card data	161	160	0.62	5.68

and intricate set of specifications to ensure their correct and safe operation across diverse functionalities (e.g., automated irrigation, crop monitoring, yield transportation and storing system). Verifying each of these specifications individually against a system model can be computationally prohibitive and time-consuming. Our proposed property refinement approach aims to alleviate this burden by identifying a minimal, yet comprehensive, subset of properties for verification.

In this section, we empirically evaluate our approach. We aim to answer the following research questions (RQ):

1) To what extent can our property refinement technique reduce the number of LTL/CTL specifications requiring explicit model checking in practice?
2) What is the computational overhead incurred by our refinement analysis process?
3) How does the approach perform as the number of properties or the complexity of formulas increases?

To address these questions, we conducted experiments using a collection of requirement documents collected from real-world systems [19] from which we derived LTL specifications. Table I shows an overview of the types of systems we considered in addition to an overview of the results. We focused our evaluation on requirement documents which are relevant for the development of a holistic smart-farming system. Specifications might include safety properties like ensuring that system access requires authentication unless a timeout occurs, formally $G((\text{system\_available} \wedge X(\text{authentication})) \vee \text{query\_timeout})$; liveness properties such as an error message eventually being displayed or data being updated, $F(\text{error\_message}) \vee \text{data\_updated}$; or invariants like the system always maintaining a backlog, $G(\text{backlog})$[1]. The algorithm has been run using Spot [20] on a Intel Ultra 9 with 64GB of RAM.

As we already hypothezied in Section VI, the efficacy of the proposed refinement methodology is maximized in the presence of significant specification redundancy and prevalent reusable subformulas. The initial partitioning of formulas into equivalence classes based on shared atomic propositions (Algorithm 1) serves as an efficient pruning mechanism. By obviating the need for CHECKREFINEMENT calls between formulas lacking common atomic propositions—a necessary condition for non-trivial semantic refinement—this step substantially reduces the number of expensive pairwise comparisons. The extent of property reduction, detailed in Table I, directly reflects the degree of inherent logical entailment within the original specification sets. Benchmarks such as S7 (89.45% reduction) and S3 (87.38% reduction) exemplify scenarios with high internal redundancy, which our approach successfully leverages to identify a compact core set of properties for verification. These results affirm that, in practice, our technique can substantially lessen the number of specifications requiring explicit verification, particularly for complex systems with overlapping requirements, hence answering RQ1. Moreover, because of the aforementioned filtering step, the completition time remains contained with the maximum reaching around 1.6 seconds. Conversely, other system types, such as the S5 benchmark, yielding only a 0.36% reduction, illustrate the methodology's limitations when specifications exhibit minimal semantic overlap. Analysis indicated a low density of shared atomic propositions across its constituent formulas, resulting in numerous small equivalence classes with limited scope for inter-formula refinement. However, in this case, the refinement analysis phase for this benchmark was computationally inexpensive (cf. refinement times in Table I), measuring as low as 5 ms for completition. Overall, the observed refinement times, averaging 0.5 seconds, suggest that the preprocessing overhead is generally modest, especially when weighed against the potential savings from avoiding numerous, potentially lengthy, individual model checking runs. This allows us to answer RQ2.

To address RQ3 regarding scalability, we evaluated our

[1]These are all derived from S2

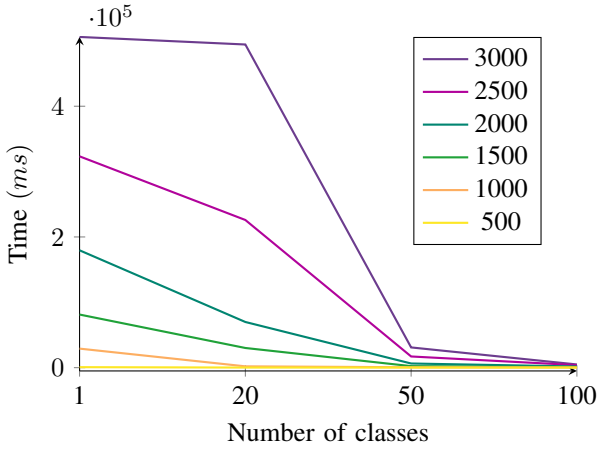

Figure 2: Analysis of the algorithm using synthetic data. The legend represents the number of properties.

refinement methodology using synthetic LTL specifications with controlled parameters, notably the number of equivalence classes. Figure 2 depicts the relationship between refinement time and the number of classes. Assuming a uniform distribution of N formulas, the runtime scales approximately quadratically with the inverse of the number of equivalence classes. This observation is consistent with our complexity analysis ($O(m^2 \cdot C_{\text{check}})$, Section VI). A greater number of distinct equivalence classes typically reduces the number of unique atomic subformulas (m) within each class. As the $O(m^2)$ term for pairwise checks is a primary driver of per-class analysis cost, smaller m values per class result in faster individual class processing, and the total time reflects the sum of these. However, non-uniform formula distribution significantly impacts performance. If a large m is concentrated in one dominant equivalence class, its refinement time will largely determine the total execution time, diminishing the scalability benefits seen with more uniform partitioning and resembling a scenario with fewer, larger effective classes.

In summary, our empirical evaluation demonstrates that the proposed property refinement methodology can significantly reduce the number of LTL specifications requiring explicit verification (RQ1). The extent of this reduction is, as hypothesized, closely tied to the inherent redundancy and semantic overlap within the original specification set, with substantial gains observed for several real-world inspired benchmarks. The computational overhead of the refinement analysis itself (RQ2) was found to be generally modest, particularly when compared to the potential savings from avoiding numerous model checking runs on large state spaces. Furthermore, our initial scalability analysis (RQ3) suggests that the approach performs well when specifications can be partitioned into a balanced set of equivalence classes, though performance can be influenced by highly skewed distributions.

Nevertheless, some interrogatives remain open. Our benchmarks, based on real-world smart farming requirements, may not capture the full diversity of LTL specification styles or system complexities across domains. Performance gains may vary with different LTL formula structures, such as deeply nested operators or varying atomic proposition patterns. Additionally, due to its greedy nature, while Algorithm 2 finds a minimal subset, it may not yield a unique minimum due to tie-breaking, though this has minimal impact on overall trends. Despite these limitations, our evaluation offers strong initial evidence for the practical value of property refinement, motivating further study across domains and its integration into broader verification toolchains.

VIII. Conclusion

Model analysis using temporal logic is a key aspect of formal verification. While refinement at a model level is a mature field within formal verification, the principles of refinement at the specification level have remained less formalized. This paper has addressed this disparity by introducing a rigorous, semantic theory of *property refinement* for temporal logics (LTL and CTL). We established its fundamental algebraic properties, providing a sound basis for a novel methodology aimed at optimizing the model checking process. Our approach systematically identifies a minimal, most-refined subset of specifications, the verification of which guarantees the satisfaction of the entire original suite, thereby enabling a more efficient verification workflows. The empirical evaluation, focusing on specifications derived from real-world system requirements, demonstrated the practical efficacy of this methodology. Significant reductions in the number of properties requiring explicit verification were achieved, particularly in scenarios with inherent specification redundancy, and the associated computational overhead for refinement analysis was found to be modest. These results underscore the potential of property refinement to alleviate the burden of verifying complex systems.

Considering the limitations identified in Sections VI and VII, future work will concentrate on extending the empirical validation to a broader range of specifications and exploring more sophisticated heuristics for the CheckRefinement subroutines for both LTL and CTL. Furthermore, alternatives to the greedy selection process for Algorithm 1 should be explored and compared, both in terms of correctness and computation overhead. Further investigation into handling highly skewed formula distributions and integrating our techniques into industrial-strength model checking toolchains also represent important next steps towards realizing the full potential of property refinement in practice.

Ultimatively, this work establishes property refinement as a key technique for optimizing formal verification and enabling the analysis of ever-more complex systems.

IX. Acknowledgements

This paper is the result of preliminary work by UNIVERSITY, on the project PROJECT_NAME, funded by grants from SPONSOR through the PROJECT_GRANT. The sole responsibility for the content lies with the authors.

REFERENCES

[1] J. Estefan, "Survey of model-based systems engineering (mbse) methodologies," *INCOSE MBSE Focus Group*, vol. 25, 01 2008.

[2] E. R. Carroll and R. J. Malins, "Systematic literature review: How is model-based systems engineering justified?" 2016.

[3] M. B. Jasser, M. Y. Said, J. Din, and A. a. abdul ghani, "A survey on refinement in formal methods and software engineering," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 8, pp. 105–112, 09 2019.

[4] A. Pnueli, "System specification and refinement in temporal logic," in *Foundations of Software Technology and Theoretical Computer Science*, R. Shyamasundar, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 1–38.

[5] R. Milner, *Communication and Concurrency*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[6] D. Park, "Concurrency and automata on infinite sequences," *Theoretical Computer Science*, vol. 25, no. 1, pp. 167–183, 1981.

[7] M. Hennessy and R. Milner, "Algebraic laws for nondeterminism and concurrency," *Journal of the ACM*, vol. 32, no. 1, pp. 137–161, 1985.

[8] N. Lynch and F. Vaandrager, "Forward and backward simulations part i: Untimed systems," *Information and Computation*, vol. 121, no. 2, pp. 214–233, 1995.

[9] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA: MIT Press, 1999.

[10] A. Mokkedem and D. Méry, "On using temporal logic for refinement and compositional verification of concurrent systems," *Theoretical Computer Science*, vol. 140, no. 1, pp. 95–138, 1995, third International Conference on Algebraic Methodology of Software Technology. [Online]. Available: https://www.sciencedirect.com/science/article/pii/030439759400206X

[11] D. Mery and D. Roegel, "Refining formal specifications to get efficient, structured and correct concurrent programs," Centre de recherche en informatique de Nancy, CNRS / Université de Nancy 1 / Université de Nancy 2 / Institut national polytechnique de Lorraine, Vandoeuvre-lès-Nancy, France, Technical Report, 1994.

[12] M. Purandare, T. Wahl, and D. Kroening, "Strengthening properties using abstraction refinement," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009, pp. 1692–1697.

[13] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 1977, pp. 46–57.

[14] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 1999.

[15] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[16] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching-time temporal logic," in *Logic of Programs, Workshop, Yorktown Heights, NY, USA, December 1980*, ser. Lecture Notes in Computer Science, vol. 131. Springer, 1981, pp. 52–71.

[17] V. Chvatal, "A greedy heuristic for the set-covering problem," *Math. Oper. Res.*, vol. 4, no. 3, p. 233–235, Aug. 1979. [Online]. Available: https://doi.org/10.1287/moor.4.3.233

[18] R. Karp, "Reducibility among combinatorial problems," vol. 40, 01 1972, pp. 85–103.

[19] A. Ferrari, G. O. Spagnolo, and S. Gnesi, "Pure: A dataset of public requirements documents," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 2017, pp. 502–505.

[20] A. Duret-Lutz, E. Renault, M. Colange, F. Renkin, A. G. Aisse, P. Schlehuber-Caissier, T. Medioni, A. Martin, J. Dubois, C. Gillard, and H. Lauko, "From Spot 2.0 to Spot 2.10: What's new?" in *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*, ser. Lecture Notes in Computer Science, vol. 13372. Springer, Aug. 2022, pp. 174–187.