Counting Models of Linear-time Temporal Logic

Bernd Finkbeiner and Hazem Torfah

Reactive Systems Group
Saarland University
66123 Saarbrücken, Germany
{finkbeiner,torfah}@cs.uni-saarland.de

Abstract. We investigate the model counting problem for safety specifications expressed in linear-time temporal logic (LTL). Model counting has previously been studied for propositional logic; in planning, for example, propositional model counting is used to compute the plan's robustness in an incomplete domain. Counting the models of an LTL formula opens up new applications in verification and synthesis. We distinguish word and tree models of an LTL formula. Word models are labeled sequences that satisfy the formula. Counting the number of word models can be used in model checking to determine the number of errors in a system. Tree models are labeled trees where every branch satisfies the formula. Counting the number of tree models can be used in synthesis to determine the number of implementations that satisfy a given formula. We present algorithms for the word and tree model counting problems, and compare these direct constructions to an indirect approach based on encodings into propositional logic.

Keywords: Model counting, temporal logic, model checking, synthesis, tree automata.

1 Introduction

Model counting, the problem of computing the *number of solutions* of a given logical formula, is a useful generalization of satisfiability. Many probabilistic inference problems, such as Bayesian net reasoning [13], and planning problems, such as computing the robustness of plans in incomplete domains [14], can be formulated as model counting problems of propositional logic. State-of-the-art tools for propositional model counting include Relsat [1] and c2d [6].

In this paper, we study the model counting problem for safety specifications expressed in *linear-time temporal logic* (LTL). LTL is the most commonly used specification logic for reactive systems [15] and the standard input language for model checking [2, 5] and synthesis tools [4, 3, 7]. Just like propositional model counting generalizes SAT, LTL model counting introduces "quantitative" extensions of model checking and synthesis. In *model checking*, model counting can be used to determine not only the existence of computations that violate the specification, but also the *number* of such *violations*. For example, in a communication system, where messages are lost (with some probability) on the channel, it is typically not necessary (or even possible) to guarantee a 100% correct transmission. Instead, the number of executions that lead to a message loss is a good indication for the quality of the implementation. In *synthesis*, model counting can be used to determine not only the existence of an implementation that satisfies the specification, but also the *number* of such *implementations*. The number of implementations of a specification is a helpful metric to understand how much room for implementation choices is left by a given specification, and to estimate the impact of new requirements on the remaining design space.

Formally, we distinguish two types of models of an LTL formula. A *word model* of an LTL formula φ over a set of atomic propositions AP is a sequence of valuations of AP such that the sequence satisfies φ. A *tree model* of an LTL formula φ over a set of atomic propositions $AP = I \cup O$, partitioned into *inputs* I and *outputs* O, is a tree that branches according to the valuations of I and that is labeled with valuations of O, such that every path of the tree satisfies φ. In order to guarantee that the number of models is finite, we consider *bounded models*, i.e., words of bounded length and trees of bounded depth. This is motivated by applications like bounded model checking [2] and bounded synthesis [8], where we look for small error paths and small implementations, respectively, by iteratively increasing a bound on the size of the model.

Since both bounded model checking and bounded synthesis are based on satisfiability checking, a natural idea to solve the model counting problem of LTL is to reduce it to the propositional counting problem: for word models, this can be done by introducing a copy of the atomic propositions for each position of the word, for tree models, by introducing a copy of the atomic propositions for each node in the tree. Unfortunately, however, this reduction quickly results in intractable propositional problems. For word models, we need a linear number of propositional variables in the bound, for tree models even an exponential number of variables. This is critical, since propositional counting is #P-complete. Current state-of-the-art model counters cannot handle more than approximately 1000-10000 propositional variables [9]. This limit is exceeded easily, for example, by a tree of depth 5. (Assuming 3 bits of input and a 3-bit encoding of the LTL formula, we need approximately 100000 variables.)

In this paper, we present a model counting algorithm with much better performance. For both word and tree models, the complexity of our algorithm is *linear* in the bound. This improvement is obtained by dynamic programming: we compute the number of models *backwards*, i.e., from the last position to the first in the case of word models, and form the leaves to the root in the case of tree models. We show that LTL formulas can be translated to word and tree automata that have exactly one run for every model. The number of runs is then computed by incrementally considering larger models and computing, for each bound, the number of models that are accepted by runs starting in a specific state.

Analyzing the complexity of this construction, it turns out that the dramatic improvement in the complexity with respect to the bound does not come for free, as our constructions are more expensive in the size of the formula, compared to the solution based on a reduction to propositional counting. In practice, however, this is not a problem, because costs in relation to the size of the formula are much more benign than costs in relation to the bound: typically, we are interested in systems with large implementations, but small specifications.

Overview. After reviewing the necessary preliminaries in Section 2, we formally define the model counting problem in Section 3. Counting algorithms for word models and tree models are presented in Sections 4 and 5, respectively.

2 Preliminaries

Transition Systems. We represent models as *labeled transition systems.* For a given finite set Υ of directions and a finite set Σ of labels, a Σ-labeled Υ-transition system is a tuple $\mathcal{S} = (S, s_0, \tau, o)$, consisting of a finite set of states S, an initial state $s_0 \in S$, a transition function $\tau : S \times \Upsilon \to S$, and a labeling function $o : S \to \Sigma$.

A *path* in a labeled transition system is a sequence $\pi : \mathbb{N} \to S \times \Upsilon$ of states and directions that follows the transition relation, i.e., for all $i \in \mathbb{N}$ if $\pi(i) = (t_i, e_i)$ then $\pi(i+1) = (t_{i+1}, e_{i+1})$ where $t_{i+1} \in \tau(t_i, e_i)$. We call the path initial if it starts with the initial state: $\pi(0) = (t_0, e)$ from some $e \in \Upsilon$. We define the set *paths*$(\mathcal{S})$ as the set of all initial paths of $\mathcal{S}$.

Specifications. We use linear-time temporal logic (LTL) [15], with the usual temporal operators Next X, Until U, and the derived operators Eventually $\Diamond$ and Globally $\Box$. LTL formulas are defined over a set of atomic propositions $AP = I \cup O$, which is partitioned into a set I of input variables and a set O of output variables. We denote the satisfaction of an LTL formula φ by an infinite sequence $\sigma : \mathbb{N} \to 2^{AP}$ of valuations of the atomic propositions by $\sigma \models \varphi$. A 2^O-labeled 2^I-transition system $\mathcal{S} = (S, s_0, \tau, o)$ satisfies φ, if for all $\pi \in$ *paths*$(\mathcal{S})$ the sequence $\sigma_\pi : i \mapsto o(\pi(i))$, where $o(s, e) = (o(s) \cup e)$, satisfies φ. In the remainder of the paper, we assume that all considered LTL specifications express safety properties. An infinite sequence $\sigma : \mathbb{N} \to 2^{AP}$ violates a safety property iff there is a prefix $\sigma' : [0, i] \to 2^{AP}$ of σ such that for all extensions $\widehat{\sigma} : \mathbb{N} \to 2^{AP}$, $\sigma' \widehat{\sigma} \not\models \varphi$. We call σ' a *bad prefix* for φ.

Universal Safety Automata. A *universal safety automaton* is a tuple $\mathcal{U} = (Q, q_0, \delta, \Sigma, \Upsilon)$, where Q denotes a finite set of states, $q_0 \in Q$ denotes the initial state, δ denotes a transition function, Σ a finite set of labels, and Υ a finite set of directions. The transition function $\delta : Q \times \Sigma \times \Upsilon \to 2^Q$ maps a state to the set of successor states reachable via a label $\sigma \in \Sigma$ and a direction $\upsilon \in \Upsilon$. A run graph of the automaton on a Σ-labeled Υ-transition system $\mathcal{S} = (S, s_0, \tau, o)$ is a directed graph $\mathcal{G} = (G, E)$ such that: The vertices $G \subseteq Q \times S$, the pair $(q_0, s_0) \in G$, and for each pair $(q, s) \in G$ there is an edge to $(q', \tau(s, \upsilon))$ for $\upsilon \in \Upsilon$ and for every $q' \in \delta(q, o(s), \upsilon)$. A transition system is accepted by the automaton if it has a run graph in the automaton.

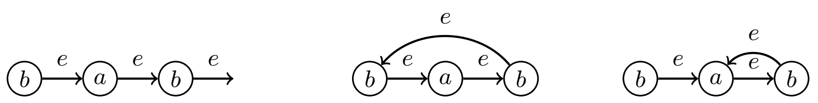


Fig. 1. A base and two word models.

For each safety property expressed as an LTL formula φ, we can construct a universal safety automaton that accepts exactly the sequences that satisfy φ. If φ has length n, then the number of states of this universal safety automaton is in $2^{\mathcal{O}(n)}$. (This can be done by translating φ into an automaton that recognizes its bad prefixes, called *fine bad prefix automaton* in [12], and dualizing this automaton.)

Bottom-up Tree Automata. Σ-labeled Υ-trees are trees where each node is labeled with a label $\alpha \in \Sigma$ and has exactly one child for every direction $\upsilon \in \Upsilon$. A *bottom-up tree automaton* is a tuple $\mathcal{T} = (T, T_F, \Delta_0, \Delta, \Sigma, \Upsilon)$ defined over Σ-labeled Υ-trees, where T is a finite set of states, $T_F \subseteq T$ denotes the set of accepting states, an initial transition relation $\Delta_0 \subseteq \Sigma \times T$ that associates a leaf node of the tree to a state of the automaton, according to the label $\alpha \in \Sigma$ of the leaf node, and the transition relation $\Delta \subseteq T^{|\Upsilon|} \times \Sigma \times T$ that determines the state labeling of a node according to the label of the node and the state labelings of the children nodes. A run of the automaton over a Σ-labeled Υ-tree is a T-labeled Υ-tree. We say that a tree is accepted by the automaton if the root of its run tree is in T_F.

3 The Model Counting Problem

A model of an LTL formula is a finite transition system. Counting the number of transition systems that satisfy a given LTL formula would not, however, be very informative, because this number is either 0 or ∞: if the formula is satisfiable, it is satisfied by some ultimately periodic model, and each unrolling of the periodic part results in a new transition system that satisfies the formula. We therefore consider *bounded models*.

We distinguish two types of bounded models, *word* and *tree* models. A *k-word model* of an LTL formula φ over $AP = I \cup O$ is a lasso sequence $\pi(0) \ldots \pi(i-1)(\pi(i), \ldots \pi(k))^\omega \in (2^O \times 2^I)^\omega$ for some $i \in \{0, ..., k\}$. We call $\pi_\perp = \pi(0) \ldots \pi(k) \in (2^O \times 2^I)^{k+1}$ the *base* of the model. Figure 1 shows two word models and their base.

A *k-tree model* of an LTL formula φ is a 2^O-labeled-2^I-transition system that forms a tree of depth k with additional loop-back transitions from the leaves (for every leaf and every direction, there is an edge to some state on the branch leading to the leaf). The tree without the loop-back transitions is the *base* of the model. Figure 2 shows two tree models and their base.

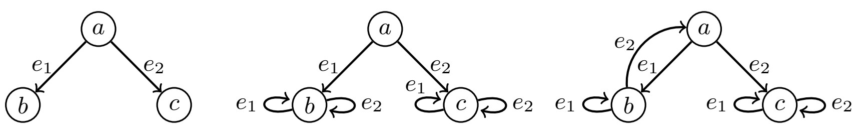


Fig. 2. A base and two tree models.

For an LTL formula φ and a bound k, the *k-word* (*k-tree*) *counting problem* is to compute the number of k-word (k-tree) models of φ.

4 Counting Word Models

We start by introducing an algorithm for counting word models of safety LTL formulas. In the next section we show how we can adapt the ideas of this algorithm in order to count tree models. For a given bound k and a safety specification φ, we construct a word automaton that accepts a finite sequence of size k if it is a base for a word model of φ. We introduce an algorithm based on the automaton that delivers the number of word models of φ.

4.1 An Automaton for Word Models

The following theorem shows that for each safety property expressed as an LTL specification φ and a bound k, we can construct a word automaton that accepts a word of maximum length k if it is a base of a word model of φ. In theorem 2 we show that the word automaton can be used to count the number of k-word models for the specification φ. Our starting point is the representation of the specification φ as a universal safety automaton. When a word model π satisfies φ, then there is a run graph of the universal safety automaton on π. In the run graph, every state s in π is mapped to a set of states in the universal automaton. This set is the set of universal states visited by π in the state s. We refer to this set as an *annotation* of s. Intuitively, our word automaton tries to reconstruct a possible annotation for each state for a given base of a word model. The loop in the word model corresponds to a suffix of the base. The annotation of this suffix is a repeating annotation in the run graph of the word model. The word automaton guesses the annotation of the loop-back state (the first state of the suffix), and checks, traversing the base *backwards*, whether (1) a repetition of the guessed annotation along the base is observed, and (2) an initial annotation is reached after having traversed the whole base (an annotation containing the initial state of the universal safety automaton). Since one base may correspond to several word models, the automaton also keeps track of the number of repetitions of the guessed annotation.

It remains to ensure that the automaton is unambiguous with respect to a word model, i.e, that every base has at most a single accepted annotation for a

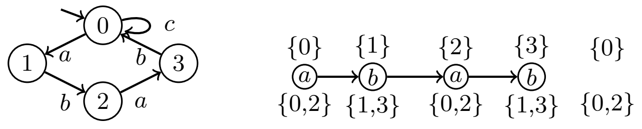


Fig. 3. On the left: a universal safety automaton; on the right: a base with two different annotations. The annotation shown above the base corresponds to a run graph of the universal automaton. The annotation shown below the base is the maximal annotation.

word model. So far, a single base might have multiple annotations. Such a situation is depicted in Figure 3. To prevent multiple annotations of the same base, the automaton only allows *maximal* annotations: in addition to the "positive" annotation, the automaton builds a "negative" annotation consisting of states of the universal automaton that must not occur in the positive annotation. All states that do not occur in the positive annotation must occur in the negative annotation. Due to the determinism of the universal safety automaton, there is only one maximal annotation for each word model over its base. Note that the maximal annotation in Figure 3 (shown below the base) includes the alternative annotation shown above the base. Also note that the maximal annotation fits two different word models, the word model with the loop labeled *abab* and the word model labeled *ab*.

Theorem 1. *Given a universal safety automaton $\mathcal{U} = (Q, q_0, \delta, \Sigma, \Upsilon)$ with n states, and a bound k, we can construct a finite word automaton $\mathcal{A}_\# = (Q_\#, Q_{0\#}, Q_{F\#}, \Delta, \Sigma, \Upsilon)$ that accepts a sequence $\sigma \in (\Sigma \times \Upsilon)^k$ if σ^{-1} is the sequence of labels of a base of a word model that is accepted by $\mathcal{U}$. The number of states of the automaton $\mathcal{A}_\#$ is in $2^{\mathcal{O}(n)}$.*

Construction: We choose $(2^Q \times \{0, ..., k\} \times (2^Q)^{n-1}) \times 2^Q \times (2^Q)^n$ to be the state space of the word automaton. A state $((\mathcal{C}, c, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1}), \mathcal{P}, \mathcal{N}_0, \ldots, \mathcal{N}_{n-1})$ is split into conjecture sets $\mathcal{C}, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1}$, which once chosen cannot be manipulated by the transition relation, and tracking sets $\mathcal{P}, \mathcal{N}_0, \ldots, \mathcal{N}_{n-1}$, which keep track of the possible state annotations for a given sequence π, starting from the conjecture annotations. The conjecture $\mathcal{C}$ denotes a loop back annotation reached in some state s on π when looping back to s. Given $\mathcal{C}$, the idea is to traverse π backwards and check whether this annotation is repeated in some state s on π. If this is the case, we point to a possible word model with a loop back to s. The counter c denotes the number of valid repetitions of $\mathcal{C}$ along π. The conjectures $\mathcal{C}_1, \ldots, \mathcal{C}_{n-1}$ are used for the maximality check. As discussed earlier, when we check a finite sequence in an inverse fashion, we may find more than one valid annotation of the universal automaton for its states. To check whether the annotation is maximal, we also compute for each state a set of negative universal states that are not allowed to be in a sequence state's annotation. For a state s in the sequence, a set $\mathcal{C}_j$ involves states that lead to a dead end in the automaton $\mathcal{U}$ in the j-th loop to s. Starting with $\mathcal{N}_0 = \emptyset$ we need to loop at most $n-1$ times to reach the largest set $\mathcal{C}_{n-1}$ of all negative universal states (the set is at most as large the set of universal states. In each loop this set either increases or we will have reached a fix-point in which all negative states are already included). An annotation is maximal if it contains all states that are not in the negative set. An initial state is a conjecture state of the form $((\mathcal{C}, 0, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1}), \mathcal{C}, \emptyset, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1})$ where $\mathcal{C}, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1} \subseteq 2^Q$.

The sets $\mathcal{P}, \mathcal{N}_0, \ldots, \mathcal{N}_{n-1}$ are computed via the transition relation Δ. Once the automaton made a choice for an initial conjecture state $((\mathcal{C}, 0, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1}), \mathcal{C}, \emptyset, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1})$, Δ becomes deterministic. For a symbol $\alpha \in \Sigma \times \Upsilon$ and a state $\Lambda_\# = ((\mathcal{C}, c, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1}), \mathcal{P}, \mathcal{N}_0, \ldots, \mathcal{N}_{n-1})$ the transition relation computes a state $\Lambda'_\# = \Delta(\Lambda_\#, \alpha) = ((\mathcal{C}, c', \mathcal{C}_1, \ldots, \mathcal{C}_{n-1}), \mathcal{P}', \mathcal{N}'_0, \ldots, \mathcal{N}'_{n-1})$ as follows. The set $\mathcal{P}'$ contains all the states of the universal automaton that lead to exactly the set $\mathcal{P}$ via the transition with α, i.e., $\max\{\mathcal{P}' \mid \bigcup_{q \in \mathcal{P}'} \delta(q, \alpha) = \mathcal{P}\}$. If such a set does not exist then there is no transition with α from this state. For the sets of maximality check Δ computes $\mathcal{N}'_i$ such that it contains all universal states that may lead via α to a state in $\mathcal{N}_i$ or have no transition with σ, i.e., $\max\{\mathcal{N}' \mid \forall q' \in \mathcal{N}'_i.\ \delta(q', \sigma) = \emptyset \ \vee\ \exists q \in \mathcal{N}_i.\ q \in \delta(q', \sigma)\}$.

A loop is found if the initial conjecture $\mathcal{C}$ is repeated i.e. $\mathcal{C} \subseteq \mathcal{P}$, and the maximality check holds. The latter is true when all positive states are in $\mathcal{P}$, i.e., $\overline{\mathcal{P}} = \mathcal{N}_{n-1}$, and for all $j < n-1$, $\mathcal{N}_j = \mathcal{C}_{j+1}$, and a fix-point for the set of negative states is reached, i.e., $\mathcal{C}_{n-1} = \mathcal{N}_{n-1}$. In this case the counter c is then incremented by one. A sequence is accepted if a state $\Lambda_\# = ((\mathcal{C}, c, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1}), \mathcal{P}, \mathcal{N}_0, \ldots, \mathcal{N}_{n-1})$ is reached after reading the last symbol σ_0, s.t., $q_0 \in \mathcal{P}$ and $c > 0$.

The unambiguity of the automaton with respect to a word model follows immediately from the maximality check and the determinism of the transition relation after having chosen the initial state. □

4.2 An Algorithm for Counting Word Models

Theorem 2. *There is a procedure that counts the number of k-word models of a safety specification expressed as an LTL formula φ in time linear in the bound k and double-exponential in the length of φ.*

Algorithm 1 describes a procedure for computing the number of word models of bases accepted by the automaton. The algorithm computes for each state of the automaton the number of bases of length i that are accepted in this state in the i-th iteration (when this state is visited in the i-th step). Ω maps each accepting state in the k-th iteration to the number of bases of length k that are accepted by the automaton. For an accepting state $q = ((\mathcal{C}, c, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1}), \mathcal{P}, \mathcal{N}_0, \ldots, \mathcal{N}_{n-1})$, a base accepted in this state has a loop annotation $\mathcal{C}$ and it is repeated c times. Thus, each base accepted in q has c word models. The number of word models is computed by summing up the number of word models in each accepting state. The algorithm traverses the automaton k times, resulting in a complexity of $\mathcal{O}(k).2^{2^{\mathcal{O}(|\varphi|)}}$.

```
$\Omega = \{(q,1) |\ q \in Q_{0\#}\}$
**for** $(i := 0, i \leq k, i{+}{+})$ **do**
    **for all** $q \in \Omega$ **do**
        **for all** $\sigma \in \Sigma \times \Upsilon$ **do**
            $\Omega(\Delta(q,\sigma)) \mathrel{+}:= \Omega(q)$

**return** $\sum_{q=((\mathcal{C},c,\mathcal{C}_1,\dots,\mathcal{C}_{n-1}),\mathcal{P},\mathcal{N}_0,\dots,\mathcal{N}_{n-1})\in Q_{F\#}} \Omega(q)\cdot c$
```

Algorithm 1: Counting with $\mathcal{A}_\#$.

5 Counting Tree Models

In this section, we introduce the counting algorithm for tree models. Our starting point is again the universal safety automaton. Similar to the case of word model counting, we guess a loop annotation and check whether the annotation is repeated when exploring the tree from its leaves upwards. However, because tree models are a composition of word models, we need to guess an annotation for each branch of the tree (for each leaf). Furthermore, as described in Section 2, a tree model must preserve the input structure of a transition system, i.e., a tree model has loop-backs from each leaf for each direction. Therefore, we have a conjecture annotation for each direction in each leaf. Traversing the tree upwards we apply then the procedure of the word case with an additional merging procedure that merges all the information received from the children in their parent tree state.

The following theorem shows that for each safety property expressed as an LTL specification φ and a bound k we can construct a bottom-up tree automaton, that accepts a tree if it is a base for a tree model of φ. Theorem 4 shows that this automaton can be used to count the number of k-tree models for the specification φ.

Theorem 3. *Given a universal safety automaton $\mathcal{U} = (Q, q_0, \delta, \Sigma, \Upsilon)$ with n states, and a bound k, we can construct a bottom-up tree automaton $\mathcal{T}_\# = (Q_\#, Q_{F\#}, \Delta_0, \Delta, \Sigma, \Upsilon)$ that accepts a Σ-labeled Υ-tree of depth k if it is a tree base of a tree model that is accepted by $\mathcal{U}$. The number of states of $\mathcal{T}_\#$ is double exponential in n.*

Construction: We choose $((2^Q \to \{\bot\} \cup \{0,\dots,k\}) \times (2^Q)^{n-1}) \times 2^Q \times (2^Q)^n$ to be the state space of $\mathcal{T}_\#$. The universal safety automaton $\mathcal{U}$ has a unique annotation for every tree model. A state $((f, \mathcal{C}_1,\dots,\mathcal{C}_{n-1}), \mathcal{P}, \mathcal{N}_0,\dots,\mathcal{N}_{n-1})$ is again split into a conjecture part $f, \mathcal{C}_1,\dots,\mathcal{C}_{n-1}$ and a tracking part $\mathcal{P}, \mathcal{N}_0,\dots,\mathcal{N}_{n-1}$. The tracking sets assign a node of a tree with the set of its positive and negative universal states. These annotations are reached from the conjecture sets of all leaves that lead upwards to this node. The conjecture part differs from the word case in the conjecture function f. The conjecture function is a partial function that maps an annotation to the number of expected repetitions along a branch of the input tree. At leaf level, the function maps the guessed annotation $\mathcal{C}$ to some number $\mu \in \{0, ..., k\}$. Moving upwards in the tree the transition relation counts down the number of repetitions of $\mathcal{C}$. In each node of the input tree the function f is a bookkeeping process for the repetitions of all the conjectures at leaf level up to this node. A node annotation is maximal if $\overline{\mathcal{P}} = \mathcal{N}_{n-1}$.

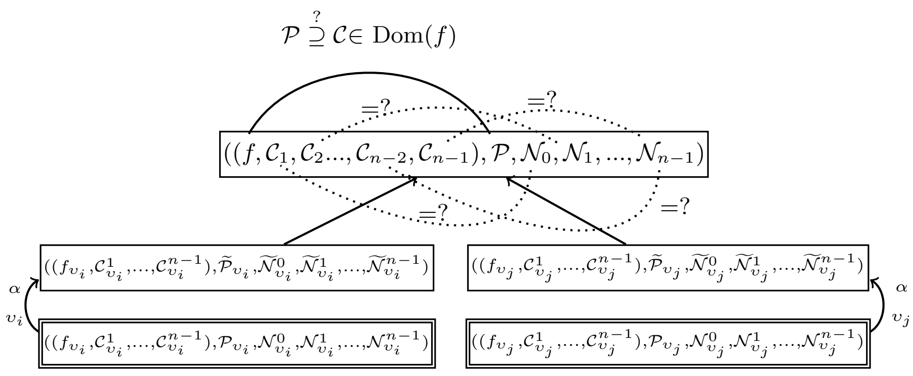
The sets $\mathcal{P}, \mathcal{N}_0, \ldots, \mathcal{N}_{n-1}$ are computed via the transition relations as follows. For each leaf state s labeled with $\alpha \in \Sigma$ the initial transition relation Δ_0 guesses an annotation $\mathcal{C}_{\upsilon_i}$ and sets $\mathcal{C}^1_{\upsilon_i}, ..., \mathcal{C}^{n-1}_{\upsilon_i}$ for each direction $\upsilon_i \in \Upsilon$. It then uses the transition relation Δ to compute the state labeling of leaf state s by computing $\Delta(\Lambda^{\upsilon_1}_{\#}, \ldots, \Lambda^{\upsilon_{|\Upsilon|}}_{\#}, \alpha)$ with $\Lambda^{\upsilon_i}_{\#} = ((f_{\upsilon_i}, \mathcal{C}^1_{\upsilon_i}, ..., \mathcal{C}^{n-1}_{\upsilon_i}), \mathcal{C}_{\upsilon_i}, \emptyset, \mathcal{C}^1_{\upsilon_i}, ..., \mathcal{C}^{n-1}_{\upsilon_i})$, where f_{υ_i} is a singleton function that maps $\mathcal{C}_{\upsilon_i}$ to some number $\mu \in \{0, ..., k\}$.

The transition relation Δ is deterministic. For states $\Lambda^{\upsilon_1}_{\#}, \ldots, \Lambda^{\upsilon_{|\Upsilon|}}_{\#}$ with $\Lambda^{\upsilon_i}_{\#} = ((f_{\upsilon_i}, \mathcal{C}^1_{\upsilon_i}, ..., \mathcal{C}^{n-1}_{\upsilon_i}), \mathcal{P}_{\upsilon_i}, \mathcal{N}^0_{\upsilon_i}, \mathcal{N}^1_{\upsilon_i}, ..., \mathcal{N}^{n-1}_{\upsilon_i})$, and a label $\alpha \in \Sigma$, the transition relation computes a state $\Delta(\Lambda^{\upsilon_1}_{\#}, \ldots, \Lambda^{\upsilon_{|\Upsilon|}}_{\#}, \alpha) = \Lambda_{\#} = ((f, \mathcal{C}_1, ..., \mathcal{C}_{n-1}),$ $\mathcal{P}, \mathcal{N}_0, \mathcal{N}_1, ..., \mathcal{N}_{n-1})$ such that, $\mathcal{C}_i = \bigcup_{\upsilon_i \in \Upsilon} \mathcal{C}^i_{\upsilon_i}$. $\mathcal{P}$ is the largest set that leads via the label α and direction υ_i to exactly the set $\mathcal{P}_{\upsilon_i}$, i.e., $\max\{\mathcal{P} \mid \bigcup_{q \in \mathcal{P}} \delta(q, \alpha, \upsilon_i) = \mathcal{P}_{\upsilon_i}\}$. If such set does not exist then there is no transition for α from this state. To compute such a set we compute for each $\mathcal{P}_{\upsilon_i}$ a set $\widetilde{\mathcal{P}}_{\upsilon_i}$ in the same fashion as in the word case. We compute then the intersection $\bigcap_i \widetilde{\mathcal{P}}_{\upsilon_i}$ and check whether the latter condition holds. Each set $\mathcal{N}_i$ must contain all universal states that may lead via α and direction υ_i to a state in $\mathcal{N}^i_{\upsilon_i}$ or have no transition with α and υ_i, i.e., $\max\{\mathcal{N}_i \mid \forall q' \in \mathcal{N}_i.\ \delta(q', \alpha, \upsilon_i) = \emptyset\ \vee\ \exists q \in \mathcal{N}^i_{\upsilon_i}.\ q \in \delta(q', \alpha, \upsilon_i)\}$. Thus, it is the union of all sets $\widetilde{\mathcal{N}}_{\upsilon_i}$ that may lead to $\mathcal{N}^i_{\upsilon_i}$ and the set $\mathcal{N}_{\not\to}$ of states that reach no state via α and any υ_i.

A new mapping f is also computed. The domain of f is the union of the domains of all f_{υ_i}. If some $\mathcal{C}$ is shared between two domains of functions f_{υ_i} and f_{υ_j}, then we require that $(f_{\upsilon_i}(\mathcal{C})) = (f_{\upsilon_j}(\mathcal{C}))$. If this condition is violated then there is no transition for α from this state. For all $\mathcal{C}$ with $f_{\upsilon_i}(\mathcal{C}) = c$, if $\mathcal{C} \subseteq P$, for all $j < n-1$, $\mathcal{N}_j = \mathcal{C}_{j+1}$, $\mathcal{C}_{n-1} = \mathcal{N}_{n-1}$, and $\overline{\mathcal{P}} = \mathcal{N}_{n-1}$, then a loop with $\mathcal{C}$ is found and we assign $f(\mathcal{C}) = c - 1$. Otherwise $f(\mathcal{C}) = f_{\upsilon_i}(\mathcal{C})$. If $c \leq 0$ then $c - 1 = \bot$. A state $((f, \mathcal{C}_1, ..., \mathcal{C}_{n-1}),\ \mathcal{P}, \mathcal{N}_0, \mathcal{N}_1, ..., \mathcal{N}_{n-1})$ is accepting if $q_0 \in \mathcal{P}$ and for all $\mathcal{C}$ in the domain of f, $f(\mathcal{C}) = 0$ (This means the guess of the number of repetitions was correct). The progress of the transition relation is depicted in Figure 4. The unambiguity of the tree automaton with respect to tree models follows from the fact that for each tree there is exactly one maximal annotation and from the determinism of the transition relation Δ. □

5.1 An Algorithm for Counting Tree Models

Theorem 4. *There is a procedure that counts the number of k-tree models of a safety specification expressed as an LTL formula φ in time linear in the bound k and triple-exponential in the length of φ.*



$\mathcal{P} \overset{?}{\supseteq} \mathcal{C} \in \mathrm{Dom}(f)$
$=?$
$=?$
$((f, \mathcal{C}_1, \mathcal{C}_2 ..., \mathcal{C}_{n-2}, \mathcal{C}_{n-1}), \mathcal{P}, \mathcal{N}_0, \mathcal{N}_1, ..., \mathcal{N}_{n-1})$
$=?$
$=?$
$((f_{v_i}, \mathcal{C}^1_{v_i}, ..., \mathcal{C}^{n-1}_{v_i}), \widetilde{\mathcal{P}}_{v_i}, \widetilde{\mathcal{N}}^0_{v_i}, \widetilde{\mathcal{N}}^1_{v_i}, ..., \widetilde{\mathcal{N}}^{n-1}_{v_i})$
$((f_{v_j}, \mathcal{C}^1_{v_j}, ..., \mathcal{C}^{n-1}_{v_j}), \widetilde{\mathcal{P}}_{v_j}, \widetilde{\mathcal{N}}^0_{v_j}, \widetilde{\mathcal{N}}^1_{v_j}, ..., \widetilde{\mathcal{N}}^{n-1}_{v_j})$
α
v_i
α
v_j
$((f_{v_i}, \mathcal{C}^1_{v_i}, ..., \mathcal{C}^{n-1}_{v_i}), \mathcal{P}_{v_i}, \mathcal{N}^0_{v_i}, \mathcal{N}^1_{v_i}, ..., \mathcal{N}^{n-1}_{v_i})$
$((f_{v_j}, \mathcal{C}^1_{v_j}, ..., \mathcal{C}^{n-1}_{v_j}), \mathcal{P}_{v_j}, \mathcal{N}^0_{v_j}, \mathcal{N}^1_{v_j}, ..., \mathcal{N}^{n-1}_{v_j})$

Fig. 4. A transition of the tree automaton over trees with directions υ_i and υ_j. The transition reads in this step a node labeled with α. The double lined states are the state labelings of the children nodes in directions υ_i and υ_j.

Algorithm 2 describes a procedure for computing the number of tree models of tree bases accepted by the automaton. The algorithm starts at the initial states computed by Δ_0. These states involve the initial conjectures for the number of expected repetitions of an initial guessed annotation. Each initial state is mapped via the function Θ to the number of expected repetitions of the initial annotation (at this level the conjecture function is defined only over one annotation). We track in each step every possible transition in Δ. A transition involves states $q_0, ..., q_{\upsilon-1}$ and a parent state q. Recall that a transition exists only if for all the shared annotations in the domains of the conjecture functions $f_0, ...f_{\Upsilon-1}$ of $q_0, ..., q_{\upsilon-1}$, the number of expected remaining repetitions is identical (this means that the initial guess was correct). For an annotation $\mathcal{C}$ shared among the domains of conjecture functions (not necessary all of them), let $c_0, ...c_h$ be initial guesses for $\mathcal{C}$ (the number of loop backs of each leaf annotated with $\mathcal{C}$). The number of possible loop combinations from those leaves is $\prod_i c_i$. In the k-th iteration each accepting state q' of $\mathcal{T}_\#$ is mapped to a function $\Theta(q)$ that defines for each annotation $\mathcal{C}$ in the domain of $f_{q'}$ the number of possible loop combinations for $\mathcal{C}$ in a tree accepted in q'. By multiplying all possible loop combination for each defined annotation we get the number of tree models of the tree accepted in q'. Finally, we sum up the results for all accepting states. The automaton is traversed k times before obtaining the final result.

6 Discussion

We have studied the model counting problem for safety specifications expressed in LTL. Counting word and tree models of LTL formulas opens up new "quantitative" versions of the classic model checking and synthesis problems for reactive systems: instead of just *checking* correctness and realizability, respectively, we

```
$\Theta : Q_{\#} \times 2^Q \to \mathbb{N}$
$Q_{0\#}$ : initial states guessed by $\Delta_0$
**Let** $q = (f_q, \mathcal{C}_q^1, \mathcal{C}_q^{n-1}, \mathcal{P}_q, \mathcal{N}_q^0, ..., \mathcal{N}_q^{n-1})$

**for all** $\sigma \in \Sigma$ **do**
  **for all** $q \in Q_{0\#}$ **do**
    **for all** $\mathcal{C} \in \mathrm{Dom}(f_q)$ **do**
      $\Theta(q, \mathcal{C}) := f_q(\mathcal{C})$

**for** $(i := 0, i \leq k, i + 1)$ **do**
  **for all** $(q_0, q_1, \ldots, q_{\Upsilon-1}, \sigma, q) \in \Delta$ **do**
    **for all** $\mathcal{C} \in \mathrm{Dom}(f_q)$ **do**
      $\Theta(q, \mathcal{C}) + := \prod_i \Theta(q_i, \mathcal{C})$        /*only if $\mathcal{C}$ is defined for $q_i$*/

**return** $\sum_{q \in Q_{F\#}} \prod_{\mathcal{C} \in 2^Q} \Theta(q, \mathcal{C})$
```

Algorithm 2: Counting with $\mathcal{T}_{\#}$.

can now judge the severity of the error by *counting* the number of error paths, and judge the specificity of the specification by *counting* the number of implementations.

While our algorithms are the first to specifically solve the model counting problem for safety specifications expressed in LTL, obvious competitors are the reduction to propositional model counting, as well as a direct enumeration of the models. As discussed in the introduction, the reduction to propositional counting is not a viable solution, because the reduction quickly leads to propositional constraints with far more than the 1000-10000 variables that can be handled by currently available model counters [9].

For k-word models, the complexity of our counting algorithm is double-exponential in the length of the LTL formula and linear in k. If the complexity in the formula were our main concern, we could do better better than this by exhaustively enumerating all words of length k: checking whether a *specific* sequence satisfies an LTL formula can be done in polynomial time (or even in NC [10]). However, the enumeration of all words takes exponential time in k, which is, for reasonable values of k, impractical. For k-tree models, the situation is similar: enumerating all trees would allow us to exploit inexpensive model checking algorithms for finite trees [11], but would result in double-exponential complexity in k, while our algorithm maintains linear complexity in k at the price of triple-exponential complexity in the length of the formula.

In future work, we plan to extend the model counting algorithms to full LTL, and to investigate the complexity for other fragments of LTL besides safety. The high complexity in the length of the formula results from the necessity to memorize information about each leaf of the tree. Fragments where this information is not needed, such as reachability properties, should therefore result in less expensive model counting algorithms.

References

1. Bayardo, R.J., Schrag, R.: Using csp look-back techniques to solve real-world sat instances. In: AAAI/IAAI. pp. 203–208 (1997)
2. Biere, A.: Bounded model checking. In: Handbook of Satisfiability, pp. 457–481. IOS Press (2009)
3. Bloem, R.P., Gamauf, H.J., Hofferek, G., Könighofer, B., Könighofer, R.: Synthesizing robust systems with RATSY. In: Association, O.P. (ed.) Proceedings First Workshop on Synthesis (SYNT 2012). vol. 84, pp. 47 – 53. Electronic Proceedings in Theoretical Computer Science (2012)
4. Bohy, A., Bruyère, V., Filiot, E., Jin, N., Raskin, J.F.: Acacia+, a tool for LTL synthesis. In: CAV. pp. 652–657 (2012)
5. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond (1992)
6. Darwiche, A.: New advances in compiling cnf into decomposable negation normal form. In: In ECAI. pp. 328–332 (2004)
7. Ehlers, R.: Unbeast: Symbolic bounded synthesis. In: Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 6605, pp. 272–275. Springer-Verlag (2011)
8. Finkbeiner, B., Schewe, S.: Bounded synthesis. International Journal on Software Tools for Technology Transfer 15(5-6), 519–539 (2013)
9. Gomes, C.P., Sabharwal, A., Selman, B.: Model counting. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 633–654. IOS Press (2009), http://dblp.uni-trier.de/db/series/faia/faia185.html#GomesSS09
10. Kuhtz, L., Finkbeiner, B.: LTL path checking is efficiently parallelizable. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2009), Part II. LNCS, vol. 5556, pp. 235–246. Springer (2009)
11. Kuhtz, L., Finkbeiner, B.: Weak Kripke structures and LTL. In: Proceedings of the 22nd international conference on Concurrency theory. pp. 419–433. CONCUR'11, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=2040235.2040272
12. Kupferman, O., Lampert, R.: On the construction of fine automata for safety properties. In: ATVA. pp. 110–124 (2006)
13. Littman, M.L., Majercik, S.M., Pitassi, T.: Stochastic boolean satisfiability. Journal of Automated Reasoning 27, 2001 (2000)
14. Morwood, D., Bryce, D.: Evaluating temporal plans in incomplete domains. In: AAAI (2012)
15. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science. pp. 46–57. SFCS '77, IEEE Computer Society, Washington, DC, USA (1977), http://dx.doi.org/10.1109/SFCS.1977.32