Synthesizing Skeletons for Reactive Systems*

Bernd Finkbeiner and Hazem Torfah

Saarland University

Abstract. We present an analysis technique for temporal specifications of reactive systems that identifies, on the level of individual system outputs over time, which parts of the implementation are determined by the specification, and which parts are still open. This information is represented in the form of a labeled transition system, which we call skeleton. Each state of the skeleton is labeled with a three-valued assignment to the output variables: each output can be true, false, or open, where true or false means that the value must be true or false, respectively, and open means that either value is still possible. We present algorithms for the verification of skeletons and for the learning-based synthesis of skeletons from specifications in linear-time temporal logic (LTL). The algorithm returns a skeleton that satisfies the given LTL specification in time polynomial in the size of the minimal skeleton. Our new analysis technique can be used to recognize and repair specifications that underspecify critical situations. The technique thus complements existing methods for the recognition and repair of overspecifications via the identification of unrealizable cores.

1 Introduction

The great advantage of synthesis is that it constructs an implementation automatically from a specification – no programming required. The great disadvantage of synthesis is that the synthesized implementation is only as good as its specification, and writing good specifications is extremely difficult.

Roughly speaking, there are two fundamental errors that can happen when writing a specification. The first type of error is to *overspecify* the system such that actually no implementation exists anymore. This type of error can be found by a synthesis algorithm (it fails!), and synthesis tools commonly assist in the repair of such errors by identifying an unrealizable core of the specification (cf. [1, 11, 12]). The second type of error is to *underspecify* the system such that not all implementations that satisfy the specification actually perform as intended. This type of error is much harder to detect. The synthesis succeeds, and even if we convince ourselves that the synthesis tool has actually chosen an implementation that performs as intended, there is no guarantee that this will again be the case when a new implementation is synthesized from the same or an extended specification.

* This work was partially funded by the European Research Council (ERC) Grant OSARES (No. 683300) and by the Deutsche Telekom Foundation.

The underlying problem is that synthesis algorithms have the freedom to resolve any underspecified behavior in the specification, and we have no way of knowing which parts of the behavior were fixed by the specification, and which parts were chosen by the synthesis algorithm.

In this paper, we introduce a new artifact that can be produced by synthesis algorithms and which provides exactly this information. We call this artifact the *skeleton* of the specification. We envision that synthesis algorithms would produce the skeleton along with the actual implementation, so that the user of the algorithm understands where the implementation is underspecified, and can, if so desired, strengthen the specification in critical areas.

A skeleton is a labeled transition system defined over three-valued sets of atomic propositions, where in each state of the skeleton an atomic proposition is either *true*, *false*, or *open*. For a given specification, the truth value of a proposition in some state of the skeleton is *open* if it can be replaced by *true* as well as by *false* without violating the specification. Consider for example the LTL formula $\bigcirc p$ for some atomic proposition p. Any transition system that satisfies the formula has truth value *true* for p in the second position of every path of the transition system. On the other hand, whether p is *true* or *false* in the initial state is not determined, either truth value would work. In this case, the skeleton would not fix a particular truth value, but rather leave the value of p in the initial state open. In a sense, the skeleton implements only those parts of the transition system that are determined by the specification.

Skeletons are useful to understand the meaning of partially written specifications. Consider, for example, an arbiter over two clients that share some resource. Each client can make a request to the source (via the inputs r_1 and r_2) and the arbiter can, accordingly, decide to give out grants via the outputs g_1 and g_2. A specification for the arbiter might begin with the property of mutual exclusion, i.e., the LTL formula $\square(\bar{g}_1 \vee \bar{g}_2)$ stating that only one of the clients should have access to the resource at a time. Figure 1 shows an implementation of this specification as a transition system and a skeleton. The transition system has a single state, and no grants are given at any time (see Figure 1(a)). The skeleton shown in Figure 1(b) reveals that all outputs are open, as indicated by the question mark. If we extend the specification with the property $\bar{g}_1 \wedge \bar{g}_2$, then the previous transition system does not need to change, because it already satisfies the extended specification. The skeleton, on the other hand, now indicates that the output in the initial state is determined. The output in subsequent states is still open (see Figure 1(c)). Extending the specification further with the property $\square(r_1 \rightarrow \bigcirc g_1)$ results in a skeleton where the responses to requests from the first client are determined, and outputs in situations where there is no request from the first client are still open (see Figure 1(e)). An implementation for this specification could be the transition system that never gives a grant to the second client (see Figure 1(d)).

We study the *model checking* and *synthesis* problems for skeletons. For a given LTL formula φ and a skeleton $\mathcal{S}$ we say that $\mathcal{S}$ is a model of the LTL formula φ, if each trace in $\mathcal{S}$ satisfies following condition: If the truth value for

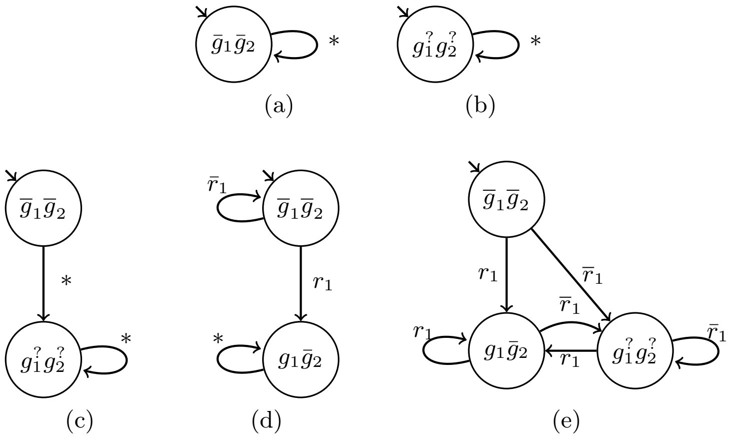


Fig. 1. Transition systems and skeletons for an arbiter specification. The symbol * denotes all possible input labels.

some proposition p in some position of the trace is open, then φ must both have a model where p is *true* at this position, and a model where p is *false* at this position. Furthermore, if the trace has truth value *true* or *false* for p at some position, then *all* models of φ map p to the truth value *true* or *false*, respectively, at this position.

We show that given an LTL formula φ we can build a nondeterministic automaton that accepts a sequence over the three-valued semantics if it satisfies the satisfaction relation described above. The automaton is of doubly-exponential size in the length of the formula φ. With this automaton, the model checking problem can be solved in EXPSPACE.

To solve the synthesis problem, we could determinize the automaton and check whether there is a skeleton for the formula, along the lines of standard synthesis [16], but this construction would be very expensive. Instead, we introduce a synthesis algorithm for skeletons based on *learning*. We show that for each LTL formula, a skeleton that models the formula defines a safety language that can be learned using the learning algorithm L^*. The algorithm can learn a skeleton for an LTL formula in time polynomial in the size of the minimal skeleton for the specification. The membership and equivalence queries of the L^* algorithm are answered by the model checking algorithm introduced in this paper.

Related Work. There is a rich body of work on the synthesis of reactive systems from logical specifications [7, 4, 10, 13, 14]. Supplemented by many works that investigated the optimization of specification for synthesis and the identification of unrealizable specification [11, 12, 1]. Multi-valued extensions of logics have been rather popular in the verification of systems, where a simple truth value is not enough to determine the quality of implementations. Chechik et. al. provide a theoretical basis for multi-valued model checking [6], where the satisfaction relation $\mathcal{M} \models \varphi$ for a model $\mathcal{M}$ and a specification φ can be multi-valued. Bruns and Godefroid experiment on multi-valued logics and show that many algorithms for multi-valued logics can be reduced to ones for two-valued logics [5]. Easterbrook and Chechik introduce a framework where multiple inconsistent models are merged according to an underlying specification given in a multi-valued logic, where the different values in the specification represent the different levels of uncertainty, priority and agreement between the merged models [9]. In comparison to all these works, we are interested in multi-valued extensions of the models themselves and in the synthesis of such models, in order to determine the amount of information that resides in a specification.

The term skeleton has been also used by Emerson and Clarke which shall not be confused with the skeletons presented here. They presented a method for the synthesis of synchronization skeletons that abstract from details irrelevant to synchronization of concurrent systems [8]. In our skeletons, we stick to the structure of transition systems and leave place holders for the underspecified details, which may then be supplemented with further steps to a complete transition system.

2 Preliminaries

Alternating Automata. We define an *alternating Büchi automaton* as a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, where Σ denotes a finite alphabet, Q denotes a finite set of states, $q_0 \in Q$ denotes a designated initial state, $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ denotes a transition function, that maps a state and an input letter to a positive boolean combination of states, and finally the set $F \subseteq Q$ of accepting states.

We define infinite words over Σ as sequence $\sigma : \mathbb{N} \to \Sigma$. A Σ-tree is a pair $(\mathcal{T}, r)$ over a set of directions D, where $\mathcal{T}$ is a prefix-closed subset of D^* and $r : \mathcal{T} \to \Sigma$ is a labeling function. The empty sequence ϵ is called the *root*. The children of a node $n \in \mathcal{T}$ are nodes $C(n) = \{n \cdot d \in \mathcal{T} \mid d \in D\}$.

A *run* of an automaton $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ on a sequence $\sigma : \mathbb{N} \to \Sigma$ is a Q-tree $(\mathcal{T}, r)$ with $r(\epsilon) = q_0$ and for all nodes $n \in \mathcal{T}$, if $r(n) = q$ then the set $\{r(n') \mid n' \in C(n)\}$ satisfies $\delta(q, \sigma(|n|))$.

A run $(\mathcal{T}, r)$ is *accepting* if for every infinite branch $n_0, n_1, \ldots$ the sequence $r(n_0)r(n_1)\ldots$ satisfies the *Büchi condition*, which requires that some state from F occures infinitely often in the sequence $r(n_0)r(n_1)\ldots$. The set of accepted words by the automaton $\mathcal{A}$ is the language of the automaton and is denoted by $L(\mathcal{A})$. An automaton is empty iff its language is the empty set.

A *nondeterministic* automaton is a special alternating automaton, where the image of δ consists only of such formulas that, when rewritten in disjunctive normal form, contain exactly one element of Q in every disjunct.

An alternating automaton is called *universal* if, for all states q and input letters α, $\delta(q, \alpha)$ is a conjunction. A universal and nondeterministic automaton is called *deterministic*.

A *Büchi* automaton is called a *safety* automaton if $Q = F$. Safety automata are denoted by a tuple (Σ, Q, q_0, δ). For safety automata, every run graph is accepting.

Safety Languages: A finite word $w = \{1, \dots, i\} \to \Sigma$ over some finite alphabet Σ is called a bad-prefix for a language $L \subseteq \Sigma^\omega$, if every infinite word $\sigma \in (\mathbb{N} \to \Sigma)$ with prefix w is not in the language L. A language $L \subseteq (\mathbb{N} \to \Sigma)$ is called a safety language, if every $\sigma \notin L$ has a bad-prefix. We denote the set of bad-prefixes for a language L by $BP(L)$. For every safety language L we can define a finite word automaton $\mathcal{B} = (Q_\mathcal{B}, Q_{\mathcal{B},0}, F_\mathcal{B}, \delta_\mathcal{B})$ that accepts the language $BP(L)$. We call $\mathcal{B}$ the bad-prefix automaton of L.

Linear-time Temporal Logic: We use Linear-time Temporal Logic (LTL) [15], with the usual temporal operators Next $\bigcirc$, Until $\mathcal{U}$ and the derived operators Eventually $\diamondsuit$ and Globally $\square$. LTL formulas are defined over a set of atomic propositions $AP = I \cup O$, which is partitioned into a set I of input propositions and a set O of output propositions. We denote the satisfaction of an LTL formula φ by an infinite sequence $\sigma \colon \mathbb{N} \to 2^{AP}$ of valuations of the atomic propositions by $\sigma \models \varphi$. For an LTL formula φ we define the language $L(\varphi)$ by the set $\{\sigma \in (\mathbb{N} \to 2^{AP}) \mid \sigma \models \varphi\}$.

Implementations: We represent implementations 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 $\mathcal{T} = (T, t_0, \tau, o)$, consisting of a finite set of states T, an initial state $t_0 \in T$, a transition function $\tau \colon T \times \Upsilon \to T$, and a labeling function $o \colon T \to \Sigma$. A *path* in $\mathcal{T}$ is a sequence $\pi \colon \mathbb{N} \to T \times \Upsilon$ of states and directions that follows the transition function, i.e., for all $i \in \mathbb{N}$ if $\pi(i) = (t_i, e_i)$ and $\pi(i+1) = (t_{i+1}, e_{i+1})$, then $t_{i+1} = \tau(t_i, e_i)$. We call a path initial if it starts with the initial state: $\pi(0) = (t_0, e)$ for some $e \in \Upsilon$. We denote the set of initial paths of $\mathcal{T}$ by $Path(T)$. For a path $\pi \in Path(T)$, we denote the sequence $\sigma_\pi \colon i \mapsto o(\pi(i))$, where $o(t, e) = (o(t) \cup e)$ by the *trace* of π. We call the set of traces of the paths of a transition system $\mathcal{T}$ the language of the $\mathcal{T}$, denoted by $L(\mathcal{T})$.

For a set of atomic propositions $AP = O \cup I$, we say that a 2^O-labeled 2^I-transition system $\mathcal{T}$ satisfies an LTL formula φ, if and only if $L(T) \subseteq L(\varphi)$, i.e., every trace of $\mathcal{T}$ satisfies φ. In this case we call $\mathcal{T}$ a model of φ.

Multi-valued Sets: A multi-valued set over an alphabet Σ and set of values Γ is a function $v \in (\Sigma \to \Gamma)$. The simplest type of multi-valued sets is the two-valued set which define the notion of sets as we know, where Σ is a set of symbols and $\Gamma = \{\bot, \top\}$, i.e., for a two-valued set v over Σ and Γ, a symbol $a \in \Sigma$ is in v if $v(a) = \top$, and not otherwise. The set of all multi-valued sets over an alphabet Σ and a set of values Γ is denoted by Γ^Σ, e.g., in the usual set notion this is the set $\{\bot, \top\}^\Sigma$ or as we know it 2^Σ for an alphabet Σ.

For a multi-valued set $v \in \Gamma^\Sigma$ and for $p \in \Sigma$ and $h \in \Gamma$ we define the multi-valued set $v' = v[p \mapsto h]$, where $v'(p) = h$ and for all $p' \in \Sigma \setminus \{p\}$, we have $v'(p') = v(p')$. For a multi-valued set $v \in \Gamma^{\Sigma}$ and for a set $\Sigma' \subseteq \Sigma$ the set $v_{\Sigma'} \in \Gamma^{\Sigma'}$ is the multi-valued set obtained by projection from Σ to Σ'.

3 Skeletons

An *open set* over an alphabet Σ is a three-valued set $v : \{\top, \bot, ?\}^{\Sigma}$, where each element $a \in \Sigma$ is either in v denoted by $v(a) = \top$, not in v denoted by $v(a) = \bot$, or it is open whether it is in the set or not, i.e., it could be one of both, denoted by $v(a) =?$. In the remainder of the paper, we denote the set $\{\top, \bot, ?\}^{\Sigma}$ by 3^{Σ}. For two open sets $v, v' \in 3^{\Sigma}$ we define the partial order $\sqsubseteq$ such that $v \sqsubseteq v'$ if and only if for all symbols $a \in \Sigma$, $v(a) \preceq v'(a)$ with respect to the lattice $\preceq = \{(\bot, \bot), (\top, \top), (\bot, ?), (\top, ?), (?, ?)\}$.

We call a sequence σ an open sequence if it is a sequence over open sets, i.e., $\sigma \in (\mathbb{N} \to 3^{\Sigma})$. For two open sequences σ and σ' we define the partial order $\sqsubseteq$ such that $\sigma \sqsubseteq \sigma'$ if for all $i \in \mathbb{N}$, $\sigma(i) \sqsubseteq \sigma'(i)$. For a sequence $\sigma \in (\mathbb{N} \to 3^{\Sigma})$ and $\Sigma' \subseteq \Sigma$ the sequence $\sigma_{\Sigma'} \in (\mathbb{N} \to 3^{\Sigma'})$ is the sequence where for all i, $\sigma_{\Sigma'}(i) = \sigma(i)_{\Sigma'}$.

We define the satisfaction relation of LTL over open sequences as follows. Given an LTL formula φ over a set of atomic propositions $AP = O \cup I$, an open sequence σ satisfies φ, denoted by $\sigma \models \varphi$, if for each sequence $\sigma' \in L(\varphi)$ that is input equivalent to σ, i.e., $\sigma_I = \sigma'_I$, we have $\sigma' \sqsubseteq \sigma$. For a fixed sequence of inputs $\varsigma \in (\mathbb{N} \to 2^I)$, there is a unique open sequence σ with $\sigma_I = \varsigma$ that satisfies φ and that is minimial with respect to the partial order $\sqsubseteq$, i.e., for all sequences $\sigma' \in (\mathbb{N} \to 3^{AP})$ with $\sigma' \models \varphi$ and $\sigma'_I = \varsigma$, we have $\sigma \sqsubseteq \sigma'$. We call such sequence a *minimal satisfying sequence*. For an LTL formula φ, we denote the set of all minimal satisfying sequences by $\min(\varphi)$.

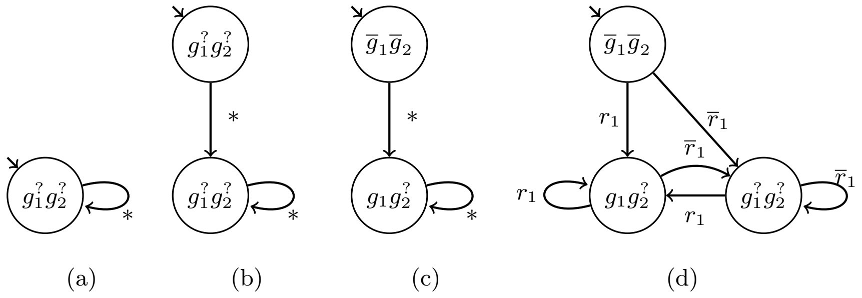
Building on the definitions of open sequences and transition systems we introduce the notion of *skeletons* of reactive systems, which are transition systems labeled with open sets from 3^O.

Definition 1 (Skeleton). *For a set $AP = O \cup I$ of atomic propositions, a* skeleton *over AP is a 3^O-labeled-2^I-transition system.*

The language of a skeleton $\mathcal{S}$ is the set of open sequences given by the set of its traces. Figure 2 shows four skeletons defined over the sets $I = \{r_1, r_2\}$ and $O = \{g_1, g_2\}$. Figures 2(a) and 2(b) both define the language $\{\sigma : \mathbb{N} \to 3^{AP} \mid \forall i. \sigma(i)(g_1) = \sigma(i)(g_2) =?\}$, i.e., for all input sequences the values of the output propositions g_1 and g_2 are open in all positions. The language of the skeleton in Figure 2(c) is the set $\{\sigma : \mathbb{N} \to 3^{AP} \mid \sigma(0)(g_1) = \sigma(0)(g_2) = \bot, \forall i > 0.\ \sigma(i)(g_1) = \top \wedge \sigma(i)(g_2) =?\}$ where the values of g_1 are fixed in all positions and for g_2 only in the first position of the sequence.[1]

We say that a skeleton $\mathcal{S}$ is a *model* of an LTL formula φ denoted by $\mathcal{S} \models \varphi$, if $L(\mathcal{S}) = \min(\varphi)$. Intuitively, for an LTL formula φ, a skeleton gives an incomplete transition system where values of atomic propositions that are not

[1] Note that skeletons have no open values for input propositions.



$\overset{?}{g}_1\overset{?}{g}_2$
$*$
(a)
$\overset{?}{g}_1\overset{?}{g}_2$
$*$
$\overset{?}{g}_1\overset{?}{g}_2$
$*$
(b)
$\overline{g}_1\overline{g}_2$
$*$
$g_1\overset{?}{g}_2$
$*$
(c)
$\overline{g}_1\overline{g}_2$
r_1
$\overline{r}_1$
$\overline{r}_1$
r_1
$g_1\overset{?}{g}_2$
r_1
$\overset{?}{g}_1\overset{?}{g}_2$
$\overline{r}_1$
(d)

Fig. 2. Skeletons over the sets $I = \{r_1, r_2\}$ and $O = \{g_1, g_2\}$

deterministically fixed by φ, are left open, i.e., they are mapped to the value ? in the open set of a state. Consider the formula $\varphi = \overline{g}_1 \wedge \overline{g}_2 \wedge \square(r_1 \rightarrow \bigcirc g_1)$. We notice that all transition systems that satisfy φ must have the label $\overline{g}_1\overline{g}_2$ in the initial state. For the rest of the transition system, the formula forces only to label a state with g_1 in case the direction(input) leading to this state contains the proposition r_1, and leaves it open on how to label the states reached by other directions, or whether to label a state with g_2 if it is reached by an input where r_1 is true (Figure 2(d)).

Building on the satisfaction relation between LTL and skeleton we investigate in the next sections the problems of model checking and synthesis of skeletons.

4 Model Checking Skeletons

We present an automata-based model checking algorithm for skeletons. Given an LTL formula φ we show that we can construct a nondeterministic Büchi automaton that recognizes the complement language $\overline{\min(\varphi)}$. Using the usual product construction, in this case, the product of the automaton and the skeleton, one can check whether the resulting automaton contains a path that simulates an accepting path in the nondeterministic automaton. If this is the case, then the language of the skeleton contains a sequence in $\overline{\min(\varphi)}$ and, thus, the skeleton is not a model for the formula φ. Using the construction of the product automaton we also show that checking whether a skeleton is a model of an LTL formula can be done in space exponential in the length of the formula.

Lemma 1. *Given an LTL formula φ we can build a nondeterministic Büchi automaton $\mathcal{N} = (3^{AP}, Q, q_0, F, \delta)$ such that $L(\mathcal{N}) = \overline{\min(\varphi)}$. The number of states of $\mathcal{N}$ is doubly-exponential in the length of φ.*

Construction. The language $\overline{\min(\varphi)}$ contains all sequences $\sigma : \mathbb{N} \rightarrow 3^{AP}$ that are not minimal satisfying open sequences for φ. These can be distinguished by two types of open sequences. The first type involves sequences σ where in some position i the truth value of a proposition $p \in AP$ is open (mapped to ?), although, in all sequences $\sigma' \in L(\varphi)$ with $\sigma_I = \sigma'_I$ the proposition p has the one same truth value (one of $\top$ or $\bot$ in all sequences) at position i. The second type are sequences σ, where in some position i a proposition p has truth value $\bot$(resp. $\top$), although, there exists another sequence $\sigma' \in L(\varphi)$ with $\sigma'_I = \sigma_I$ and $\sigma'(i)(p) = \top$(resp. $\bot$). The latter case also subsumes the case of sequences $\sigma \in (\mathbb{N} \to 2^{AP})$ with $\sigma \notin L(\varphi)$.

We construct a Büchi automaton $\mathcal{N} = (3^{AP}, Q, q_0, F, \delta)$ that accepts an open sequence σ if and only if $\sigma \notin \min(\varphi)$. The automaton is composed of two nondeterministic Büchi automata $\mathcal{N}_1 = (3^{AP}, Q_1, q_{0,1}, F_1, \delta_1)$ and $\mathcal{N}_2 = (3^{AP}, Q_2, q_{0,2}, F_2, \delta_2)$, one for each of the sequence types mentioned above. We define the automaton as $\mathcal{N} = \mathcal{N}_1 \vee \mathcal{N}_2$, where $Q = \{q_0\} \cup Q_1 \cup Q_2$, $F = F_1 \cup F_2$ and $\delta = \{(q_0, a, \delta_1(q_{0,1}, a) \vee \delta_2(q_{0,2}, a)) \mid a \in 3^{AP}\} \cup \delta_1 \cup \delta_2$

Automaton $\mathcal{N}_1$ accepts a sequence $\sigma \in (\mathbb{N} \to 3^{AP})$ if σ has a position i where an atomic proposition $p \in AP$ is incorrectly marked as open. The automaton $\mathcal{N}_1$ can be constructed as follows:

Let $\mathcal{U}_1 = (2^{AP}, Q_1^{\mathcal{U}}, q_{0,1}^{\mathcal{U}}, F_1^{\mathcal{U}}, \delta_1^{\mathcal{U}})$ be a universal Büchi automaton for the formula $\neg\varphi$. We extend the automaton $\mathcal{U}_1$ to another universal Büchi automaton $\mathcal{U}_1^*$ over an extended alphabet $\{\top, \bot, ?, *_\top, *_\bot\}^{AP}$. We make use of the values $*_\top$ and $*_\bot$ to encode in the input sequence whether a mapping to ? is wrong, and whether it is wrong when replacing ? by $\top$ or by $\bot$. We define $\mathcal{U}_1^* = (\{\top, \bot, ?, *_\top, *_\bot\}^{AP}, Q_1^*, q_{0,1}^*, F_1^*, \delta_1^*)$ over two copies of the automaton $\mathcal{U}_1$(denoted by the numbers 1 and 2) where $Q_1^* = Q_1^{\mathcal{U}} \times \{1,2\}$, $q_{0,1}^* = (q_{0,1}^{\mathcal{U}}, 1)$, $F_1^* = F_1^{\mathcal{U}} \times \{1,2\}$. The transition function δ_1^* is given by the union of the following sets:

- $\{((q,h), v, \delta_1^{\mathcal{U}}(q,v)_{\{q' \in Q_1^{\mathcal{U}}/(q',h)\}}) \mid h \in \{1,2\}, \forall p \in O. v(p) \in \{\top, \bot\}\}$
  where in both copies of the automaton $\mathcal{U}_1$, transitions over symbols v with no open values remain in the same copy and follow the structure of the transition relation $\delta_1^{\mathcal{U}}$ of $\mathcal{U}_1$. The operation $\{q' \in Q_1^{\mathcal{U}}/(q',h)\}$ substitutes every appearance of a state q' in $\delta_1^{\mathcal{U}}(q,v)$ by a state (q',h) from Q_1^*.
- $\{((q,h), v, (\delta_1^{\mathcal{U}}(q, v[p \mapsto \top]) \wedge \delta_1^{\mathcal{U}}(q, v[p \mapsto \bot]))_{\{q' \in Q_1^{\mathcal{U}}/(q',h)\}}) \mid$
  $h \in \{1,2\}, p \in O, v(p) = ?, \}$
  universal transitions for symbols where a proposition p has an open truth value imitating transitions for both truth values $\top$ and $\bot$ for p.
- $\{((q,1), v, \delta_1^{\mathcal{U}}(q, v[p \mapsto \top])_{\{q' \in Q_1^{\mathcal{U}}/(q',2)\}}) \mid p \in O, v(p) = *_\top\}$
  when we guess at some position i that an open truth value for a proposition p is wrong, and it is wrong when replacing it by $\top$ we follow the transition $\top$ to the second copy of $\mathcal{U}_1$ in which $?, *_\bot$ and $*_\top$ are treated equivalently. This helps to check, whether replacing ? by $\top$ results in accpeting run in $\mathcal{U}_1$, which means that at position i the truth value $\top$ violates the property φ, and thus it cannot be open at the that point.
- $\{((q,1), v, \delta_1^{\mathcal{U}}(q, v[p \mapsto \bot])_{\{q' \in Q_1^{\mathcal{U}}/(q',2)\}}) \mid p \in O, v(p) = *_\bot\}$
  which introduce transitions that involve the dual case of $*_\top$.
- $\{((q,2), v, (\delta_1^{\mathcal{U}}(q, v[p \mapsto \top]) \wedge \delta_1^*(q, v[p \mapsto \bot]))_{\{q' \in Q_1^{\mathcal{U}}/(q',2)\}}) \mid$
  $p \in O, v(p) \in \{*_\bot, *_\top\}\}$
  these transitions make sure that when moving to copy 2 of $\mathcal{U}_1$, values $*_\top$

and $*_\bot$ are treated equally to ?, because after guessing that a ? is wrong it must be wrong for all continuations.

In order to obtain the desired automaton $\mathcal{N}_1$ over the alphabet 3^{AP} we first transform the automaton $\mathcal{U}_1^*$ to a nondeterministic automaton $\mathcal{N}_1^*$ with $L(\mathcal{U}_1^*) = L(\mathcal{N}_1^*)$ using a subset construction. This is necessary in order to merge all transitions $*_\bot$ at one level into one state. The same holds also for transitions $*_\top$. In this way, we can check whether at some position in a sequence a value ? is wrong by checking all possible branches of the automaton $\mathcal{U}_1^*$ at that level. The automaton $\mathcal{N}_1^*$ can be transformed now to the desired automaton $\mathcal{N}_1$ by projecting every transition label with values in $\{*_\top, *_\bot\}$ to a label $v' \in 3^{AP}$ such that for every $p \in O$, if $v(p) = *_\top$ or $v(p) = *_\bot$ then $v'(p) =?$.

The size of the automaton $\mathcal{U}_1$ is exponential in the length of φ using the transformation of LTL formulas into alternating Büchi automata [17], and then using a subset construction. The transformation to $\mathcal{U}_1^*$ from $\mathcal{U}_1$, and to $\mathcal{N}_1$ from $\mathcal{N}_1^*$ are both polynomial, and exponential from $\mathcal{U}_1^*$ to $\mathcal{N}_1^*$. Thus, the size of $\mathcal{N}_1$ is doubly-exponential in the length of φ.

In a similar way, we can construct the automaton $\mathcal{N}_2$. Automaton $\mathcal{N}_2$ accepts a sequence $\sigma \in (\mathbb{N} \to 3^{AP})$ if a proposition $p \in AP$ is incorrectly mapped to $\top$ or $\bot$. Starting with the alternating Büchi automaton for the formula φ, we extend the alphabet with symbols $*_\top$ and $*_\bot$ and build an automaton $\mathcal{U}_2^* = (\{\top, \bot, ?, *_\top, *_\bot\}^{AP}, Q_2^*, q_{0,2}^*, F_2^*, \delta_2^*)$. Whenever we read a symbol v where some $p \in O$ is mapped to $*_\top(*_\bot)$, the automaton follows the transition for $v(p) = \bot(\top)$. After turning $\mathcal{U}_2^*$ to a nondeterministic automaton and projecting, a label v is replaced by a label v' such that for every $p \in O$, if $v(p) = *_\top$ or $v(p) = *_\bot$ then $v'(p) = \top$ or $v'(p) = \bot$, respectively. The automaton $\mathcal{N}_2$ is doubly-exponential in the length of φ.

Proof. Let $\sigma \in (\mathbb{N} \to 3^{AP})$. We distinguish three cases:

- $\sigma \in \overline{\min(\varphi)}$ and for some i and some $p \in O$, the mapping $\sigma(i)(p) =?$ is wrong. We assume, w.l.o.g., that for all $\sigma' \in L(\varphi)$ with $\sigma_I = \sigma'_I$, that $\sigma'(i)(p) = \top$, and that i is the first position for which $\sigma(i)(p) =?$ is wrong. A run of the automaton $\mathcal{N}_1$ over σ is a sequence $r \in (\mathbb{N} \to 2^{Q_1^*})$. Let $r = X_0X_1...$ be the run of the automaton $\mathcal{N}$ on σ, where $X_0 = \{q_{0,1}^*\}$, and up to the position i the run follows for each mapping to ? the transitions in $\mathcal{N}_1$ that were transitions for mappings to ? in the automaton $\mathcal{N}_1^*$ before the projection, i.e., all sets X_j with $j \leq i$ contain only states $(q, 1)$ from Q_1^*, where $q \in Q_1^{\mathcal{U}}$. In the position i, where the mapping to ? is incorrect, the run follows the transition with ? in state X_i of $\mathcal{N}_1$ that can be mapped to a transition $*_\bot$ in the automaton $\mathcal{N}_1^*$ which moves to a set X_{i+1} with only states $(q, 2)$ from Q_1^*, i.e., the transition that checks whether replacing ? at i with $\bot$ always leads to rejecting states for possible instantiations of upcoming ?. As $\mathcal{U}_1^*$ is built from copies of the automaton $\mathcal{U}_1$ for the formula $\neg\varphi$, following the transition for $*_\bot$ means replacing at position i the value ? with $\bot$, which can only lead to rejecting runs, because the automaton $\mathcal{U}_1$ accepts no sequence where p is mapped to value $\bot$ at position i.

– $\sigma \in \overline{\min(\varphi)}$ and for some i and some $p \in O$, $\sigma(i)(p)$ is incorrectly mapped to $\top$ or to $\bot$. With the same argumentation of the last case over the structure of the automaton $\mathcal{N}_2$ the claim can be proven.
– $\sigma \in \min(\varphi)$. In this case, for each position i, for each proposition $p \in O$ such that $\sigma(i)(p) =?$, and for each instantiation of ? for p in position i, there are instantiations for all other ? values in σ and for all propositions such that the resulting sequence $\sigma' \in (\mathbb{N} \rightarrow 2^{AP})$ is in $L(\varphi)$. Let $r = X_0X_1...$ be a run of $\mathcal{N}_1$ on σ. If r follows all transitions for a mapping to ? that correspond to a transition for the value ? in $\mathcal{N}_1^*$. In this case, all sets X_j for $j \geq 0$ have states $(q, 1)$ of $\mathcal{U}_1^*$ where $q \in Q_1^{\mathcal{U}}$ and the run is not accepting, because the run simulates a universal run tree in $\mathcal{U}_1^*$ with at least one non-accepting branch, because there is an instantiation for σ that is a model of φ. If at any point, then run r takes a transition for some mapping to ? that corresponds to a transition $*_\bot$ or $*_\top$ in the automaton $\mathcal{N}_1^*$, then the run cannot be accepting, otherwise there is a mapping to ? for some proposition $p \in O$ in some position in σ for which all other ? in σ cannot be instantiated appropriately in order to get a model in σ.
  In a similar way we can also prove that $\mathcal{N}_2$ has no accepting run for σ.

□

To check whether a skeleton $\mathcal{S}$ is a model for a given LTL formula φ we compute the product $\mathcal{P} = \mathcal{S} \times \mathcal{N}$ where $\mathcal{N}$ is nondeterministic Büchi automaton with $L(\mathcal{N}) = \overline{\min(\varphi)}$ constructed in Lemma 1. If $\mathcal{P}$ contains a path that simulates an accepting path in $\mathcal{N}$, then $\mathcal{S}$ has a path that violates the property φ, i.e., there is a sequence in the language $L(\mathcal{S})$ that is not in $\min(\varphi)$.

Instead of constructing the product automaton $\mathcal{P}$ one can also guess a run in $\mathcal{P}$ and check whether it is accepting[2]. Based on this idea, the complexity of model checking skeleton is given by the following theorem.

Theorem 1. *Checking whether a skeleton $\mathcal{S}$ is a model for an LTL formula φ is in* EXPSPACE.

5 Synthesis of Skeletons

For a set of atomic propositions $AP = I \cup O$, to check whether there is 2^O-labeled 2^I-transition system $\mathcal{T}$ that satisfies a given LTL formula φ, one would construct a deterministic ω-automaton $\mathcal{D}$ (for example a parity automaton) with $L(\mathcal{D}) = L(\varphi)$, interpret the automaton as a tree automaton over trees with labels from 3^O and directions from 2^I and check its emptiness. In case, the language of the automaton is not empty the procedure returns a transition system $\mathcal{T}$ that models the formula φ. In the same fashion, we can construct a deterministic ω-automaton for the language $\min(\varphi)$ (for example by determinizing the automaton from Lemma 1) and check whether there is a skeleton that is a model

[2] This follows the idea of the PSPACE model checking algorithm for LTL over transition systems [3]

for φ by performing an emptiness check over tree automaton interpretation of the deterministic automaton.

The deterministic automaton is very expensive to construct (triple exponential in the formula φ). Instead, we show that we can avoid this construction of the large deterministic automaton using learning. In comparison to transition systems, given an LTL formula, we show that it has a unique minimal skeleton that models the formula. The language of the skeleton is a safety language, and thus, can be characterized by a bad-prefix automaton, which is a finite word automaton. We use the learning algorithm L^* to learn the deterministic bad-prefix automaton [2], which can be easily transformed to a skeleton that models the formula. The learning algorithm learns the skeleton in time polynomial in the size of the minimal skeleton.

5.1 Learning Skeletons

In the following we present an algorithm for learning skeletons of LTL formulas. Our algorithm is based on the L^* algorithm for learning deterministic finite automata introduced by Dana Angluin [2]. The setting of the L^* algorithm involves two key actors, the *learner* and the *teacher*. The learner tries to learn a language known to the teacher by learning a minimal deterministic finite word automaton for the language. The interaction between the learner and the teacher is driven by two types of queries: *membership queries*, where the learner asks whether a particular word is in the language, and *equivalence queries*, to check whether a learned deterministic finite automaton indeed defines the language to be learned. Here, the teacher responds either with a "yes" or with a counterexample, which is a word in the symmetric difference of the language of the learned automaton and the actual language. A teacher is called *minimally adequate*, if she can answer membership and equivalence queries.

Theorem 2. [2] *Given a minimally adequate Teacher for an unknown regular language L, we can construct a minimal finite word automaton that accepts L, in time polynomial in the number of states of the automaton and the length of the largest counterexample returned by the teacher.*

For an LTL formula φ we show that the language of a skeleton that satisfies φ is a safety language. This can be characterized by a language over finite words, namely the language of bad-prefixes. The L^* algorithm can learn a finite automaton for the language of bad-prefixes, which in turn can then be transformed to a skeleton for the property φ.

Lemma 2. *For an LTL formula φ, the language* $\min(\varphi)$ *is a safety language.*

Proof. We show that every $\sigma \in \overline{\min(\varphi)}$ has a bad-prefix. We distinguish two cases for σ:

– There is a point i in σ and a proposition p such that $\sigma(i)(p) = \top(\text{or } \bot)$ and there is a sequence $\sigma' \in L(\varphi)$ with $\sigma_I = \sigma'_I$ and $\sigma'(i)(p) = \bot(\text{or}\top)$. Thus, any finite sequence $v_0 \dots v_i \in (3^{AP})^*$ with $(v_0 \dots v_i)_I = (\sigma(0) \dots \sigma(i))_I$ and $v_i(p) \neq ?$ is a bad-prefix for $\min(\varphi)$.

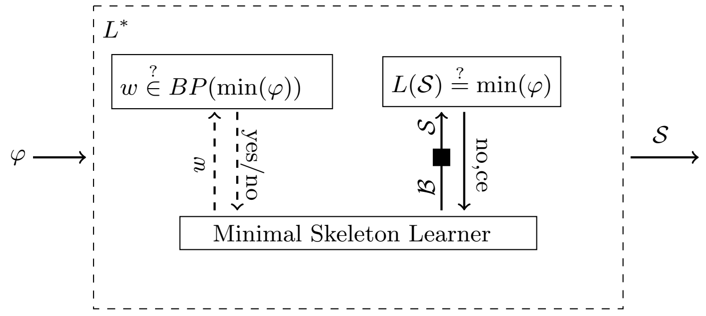


Fig. 3. A modified L* for learning minimal skeletons of LTL formulas

– There is a point i in σ and a proposition p such that $\sigma(i)(p) =?$ and for all $\sigma' \in L(\varphi)$ with $\sigma_I = \sigma'_I$ we have $\sigma'(i)(p)$ is solely $\top$ or solely $\bot$. In this case, every finite sequence $v_0 \dots v_i \in (3^{AP})^*$ with $(v_0 \dots v_i)_I = (\sigma(0) \dots \sigma(i))_I$ and $v_i(p) =?$ is a bad-prefix for $\min(\varphi)$.

□

From the last lemma we deduce, that a skeleton $\mathcal{S}$ for an LTL formula φ can be seen as a safety automaton that accepts the language of minimal satisfying open sequences for φ. In particular, there is a bad-prefix automaton $\mathcal{B}$ that accepts the language of bad-prefixes of the language $\min(\varphi)$.

We use the L* algorithm to learn a deterministic bad-prefix automaton for the language $\min(\varphi)$. Figure 3 shows a high level flow graph of the learning algorithm[3]. The learner poses a series of membership questions before making a conjecture about the bad-prefix automaton. With a membership query the learner asks whether a finite word $w \in (3^{AP})^*$ is a bad-prefix for $\min(\varphi)$. If w is a bad-prefix then the teacher returns *yes*, and *no* otherwise. The equivalence queries allow the learner to check whether a skeleton $\mathcal{S}$ is correct, i.e., $L(S) = \min(\varphi)$. The teacher either confirms the automaton or returns a counterexample to the learner. The latter is either a bad-prefix that is not rejected by $\mathcal{B}$ or word $w \in (3^{AP})^*$ that is not a bad-prefix for $\min(\varphi)$ yet is in the language of $\mathcal{B}$. The black box shown in Figure 3 between the bad-prefix automaton and a skeleton, is a check whether the safety language characterized by the bad-prefix automaton can be represented by a skeleton. We will refer to this check as the *output consistency check* and will explain it later in more detail.

The skeleton returned by the learning procedure is minimal and it is unique.

Lemma 3. *For each LTL formula φ there is a unique (up to isomorphism) minimal skeleton $\mathcal{S}$ such that $\mathcal{S} \models \varphi$.*

Proof. Let $\mathcal{S} = (S, s_0, \tau, o)$ and $\mathcal{S}' = (S', s'_0, \tau', o')$ be two minimal skeletons for φ, i.e, $|S| = |S'| = c$ and there is no skeleton $\mathcal{S}'' = (S'', s''_0, \tau'', o'')$ for φ with $|S''| < c$. We show that $\mathcal{S}$ and $\mathcal{S}'$ define the same skeleton up to isomorphism.

[3] For more details on the L* algorithm we refer the reader to [2].

Let $\beta = \{(s, s') \in S \times S' \mid \forall \sigma_I \in (2^I)^*.\ \tau^*(s0, \sigma_I) = s \leftrightarrow \tau'^*(s'_0, \sigma_I) = s'\}$. The relation β is bijective because τ^* and τ'^* are both functional and complete. Thus, there is a one-to-one mapping between the states of $\mathcal{S}$ and those of $\mathcal{S}'$, and for each $(s_1, i, s_2) \in \tau$ we have $(\beta(s_1), i, \beta(s_2)) \in \tau'$. For each $(s, s') \in \beta$ it is also the case that $o(s) = o'(s')$, otherwise, there is an input sequence that distinguishes a trace in $\mathcal{S}$ from the corresponding one in $\mathcal{S}'$, which contradicts the assumption that $L(\mathcal{S}) = L(\mathcal{S}')$. This implies that $\mathcal{S}$ is isomorphic to $\mathcal{S}'$. $\square$

In the next sections we show how membership and equivalence queries can be solved algorithmically.

5.2 Membership Queries

In this section we show that using the ideas of the automaton presented in Lemma 1 we can check whether a word is a bad-prefix in space exponential in the length of φ.

Theorem 3. *Given an LTL formula φ and a finite word $w \in (3^{AP})^*$, checking whether w is a bad-prefix for $\min(\varphi)$ is in* EXPSPACE.

Proof. A finite word $w \in (3^{AP})^*$ is a bad-prefix for $\min(\varphi)$ if $w = w_0 \dots w_n$ has a prefix and there is a sequence of input values ς and no sequence $\sigma : \mathbb{N} \to 3^{AP}$ with $\sigma_I = \varsigma$ can extend w to a sequence in $w \cdot \sigma \in \min(\varphi)$. Let $\mathcal{U} = (\Sigma, Q, q_0, \delta, F)$ be a universal Büchi automaton such that $L(\mathcal{U}) = L(\neg\varphi)$. The idea is to iteratively construct a run of the automaton $\mathcal{U}$ and check if the run is accepting (remember that a run of $\mathcal{U}$ is Q-tree). Given the input word w, we first guess which position i of w contains a wrong mapping and compute the set of states of the run tree over $w_0 \dots w_i$ reached at this position. Then, we compute the set of states reached via choosing the transition for which the guessed position i is wrong. Form here on, we guess the next input and branch universally for all valuations of the output propositions, and compute the next set of reached states. This is repeated $2^{|Q|}$ times (At latest at position $2^{|Q|}$ we reach a set of states, that was seen before and enter a loop in the run). If during the procedure a valid accepting configuration of the universal automaton was guessed, then we have found a sequence of inputs ς for which no σ with $\sigma_I = \varsigma$ extends the prefix of $w_0 \dots w_i$ to a sequence in $\min(\varphi)$. Thus, w is a bad-prefix for $\min(\varphi)$. In each step we only need to remember the currently reached set of states of $\mathcal{U}$, and whether we have seen an accepting configuration of $\mathcal{U}$. Furthermore, the number of iteration can be encoded in binary and is polynomial in the size of $\mathcal{U}$, which in turn is exponential in the length of φ. $\square$

5.3 Equivalence Queries

We move now to equivalence queries. To check whether a skeleton is a model for a formula φ we apply the model checking algorithm presented in Section 4. The learning algorithm first constructs a bad-prefix automaton for the language

$\min(\varphi)$. We show that this automaton can be turned into a safety automaton for $\min(\varphi)$ on which we can simulate a skeleton for φ. In case we cannot simulate the skeleton on top of the safety automaton, then there is no skeleton that models the formula φ.

Lemma 4. *Given a deterministic bad-prefix automaton $\mathcal{B}$ for a safety property φ, we can construct a deterministic safety automaton $\mathcal{S}$ for φ in time linear in the size of $\mathcal{B}$.*

Construction. Let $\mathcal{B} = (\Sigma, Q, q_0, F, \delta)$ be a bad-prefix automaton for some property φ and we assume it is complete. We construct a safety automaton $\mathcal{S} = (\Sigma, Q', q'_0, \delta')$ for φ by first removing all states in F and then by iteratively removing all resulting sink states in the automaton.

Remark 1. Note that if $\mathcal{B}$ is minimal, so is $\mathcal{S}$.

Before we move on to the construction we consider following fact about skeletons and the language $\min(\varphi)$ for some formula φ. Let $AP = O \cup I$ be the set of atomic propositions. Let $\mathcal{S} = (S, s_0, \tau, o)$ be a skeleton that models the formula φ. Let $\pi_1 = (s_0, i_1)(s_1, i_1)\ldots$ and $\pi_2 = (s_0, i_1)(s_1, i_2)\ldots$ be paths in $\mathcal{S}$ where $s_0, s_1 \in S$ and $i_1, i_2 \in I$. Then, both sequences $\sigma_{\pi_1} = (o(s_0) \cup i_1)(o(s_1) \cup i_1)\ldots$ and $\sigma_{\pi_2} = (o(s_0) \cup i_1)(o(s_1) \cup i_2)\ldots$, must be in the set $\min(\varphi)$, otherwise $\mathcal{S}$ is not a model of φ. This means, if the language $\min(\varphi)$ contains sequences $(o_1 \cup i_1)(o_2 \cup i_1)\ldots$ and $(o_1 \cup i_1)(o'_2 \cup i_2)\ldots$ with $o_2 \neq o'_2$ then there is no skeleton that models φ, because $\min(\varphi) = L(\mathcal{S})$ and both traces cannot be trace of the skeleton at the same time.

Definition 2 (Output Consistent). *For a set of atomic propositions $AP = O \cup I$, a safety automaton $\mathcal{A} = (3^{AP}, Q, q_0, \delta)$ is output consistent, if for each state $q \in Q$ there is a unique mapping $v \in \{\bot, \top, ?\}^O$ and for all transitions $(q, v', q') \in \delta$, $v'(p) = v(p)$ for all propositions $p \in O$.*

Lemma 5. *Given an LTL formula φ, if there is an output consistent safety automaton $\mathcal{A}$ for the language $\min(\varphi)$, we can transform $\mathcal{A}$ to skeleton $\mathcal{S}$ that models φ. The size of $\mathcal{S}$ is equal to the size of $\mathcal{A}$.*

Construction. Let φ be an LTL formula and let $\mathcal{A} = (3^{AP}, Q, q_0, \delta)$ be an output consistent safety automaton for the language $\min(\varphi)$ constructed from a deterministic bad-prefix automaton as in Lemma 4. Let $Q = \{q_0, q_1 \ldots q_n\}$. We can construct a skeleton $\mathcal{S} = (S, s_0, \tau, o)$, where $S = \{s_0, \ldots, s_n\}$ and $o(s_i) = X \cap O$ for $(q_i, X, q') \in \delta$ for some $q' \in Q$, and $(s_i, Y, s_j) \in \tau$ for $Y \subseteq I$ when $(q_i, o(s_i) \cup Y, q_j) \in \delta$. The skeleton $\mathcal{S}$ models φ, because it simulates the language of $\mathcal{A}$.

Lemma 6. *Given a formula φ, if an output consistent safety automaton $\mathcal{A}$ with $L(\mathcal{A}) = \min(\varphi)$ is minimal then the skeleton $\mathcal{S}$ extracted form $\mathcal{A}$ is also minimal.*

Proof. This follows from the fact that we can use the reverse of the construction presented in Lemma 5 to construct the safety automaton from the skeleton. Assume $\mathcal{S}$ was not minimal, then there is a skeleton $\mathcal{S}'$ with less number of states. This one, however, can be transformed backwards to a output consistent automaton of same size, which contradicts the assumption. $\square$

Once we obtain a candidate skeleton, we check whether the skeleton is a model of the formula using the model checking algorithm presented in Section 4. If the skeleton is not a model, the algorithm returns a counterexample, which is a lasso-shaped trace in the candidate skeleton. As this trace must contain a bad-prefix, we can iteratively check all prefixes of the trace using membership queries until we reach the (shortest) bad-prefix.

Using the results presented in Theorem 1 (Equivalence query checking is in EXPSPACE), Theorem 2 (L^* learns a minimal bad-prefix automaton in polynomial time in the size of the minimal automaton), Theorem 3 (Membership checking is in EXPSPACE), Lemma 2 (The language $\min(\varphi)$ can be characterized by a finite automaton), Lemma 3 (The minimal skeleton is unique), Lemma 5 (The safety automaton is a skeleton), and Lemma 6, we can conclude now with following theorem.

Theorem 4. *Given an LTL formula φ, we can construct a skeleton $\mathcal{S}$ that models φ in time polynomial in the size of the minimal skeleton of φ.*

6 Conclusion

We have presented an analysis technique for temporal specifications of reactive systems that identifies, on the level of individual system outputs over time, which parts of the implementation are determined by the specification, and which parts are still open. Based on the algorithms developed in this paper, a synthesis tool can represent this information in the form of a skeleton for the reactive system. Skeletons are more informative than conventional transition systems in identifying critical situations that are still underspecified.

Our automaton-based model checking algorithm for skeletons also serves as the teaching oracle in the learning-based synthesis algorithm. The learning algorithm L^* can be used to synthesize minimal skeletons because skeletons define safety languages, which can be characterized by a unique minimal bad-prefix automaton. Once the automaton is learned, it can directly be transformed into a skeleton for the specification. The skeleton is minimal and can be constructed in time polynomial in the number of states of the skeleton.

In the development of a reactive system, skeletons can be seen as an intermediate step between the specification of the system and its implementation. In future work, we plan to investigate this aspect further, by exploring an incremental development process, where the refinement of the specification is guided by the identification of underspecified situations through the skeletons synthesized from the intermediate specifications.

References

1. Rajeev Alur, Salar Moarref, and Ufuk Topcu. Counter-strategy guided refinement of GR(1) temporal logic specifications. In *Formal Methods in Computer-Aided Design, FMCAD 2013*, pages 26–33. IEEE, 2013.
2. Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, November 1987.
3. Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
4. Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Saar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, May 2012.
5. Glenn Bruns and Patrice Godefroid. Model checking with multi-valued logics. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004. Proceedings*, pages 281–293. Springer Verlag, 2004.
6. Marsha Chechik, Benet Devereux, Steve Easterbrook, and Arie Gurfinkel. Multi-valued symbolic model-checking. *ACM Trans. Softw. Eng. Methodol.*, 12(4):371–408, October 2003.
7. Alonzo Church. Logic, arithmetic, and automata. In *Proc. Internat. Congr. Mathematicians (Stockholm, 1962)*, pages 23–35. Inst. Mittag-Leffler, Djursholm, 1963.
8. Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, UK, 1982. Springer-Verlag.
9. Steve Easterbrook and Marsha Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 411–420. IEEE Computer Society, 2001.
10. Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *International Journal on Software Tools for Technology Transfer*, 15(5-6):519–539, 2013.
11. Robert Könighofer, Georg Hofferek, and Roderick Bloem. Debugging unrealizable specifications with model-based diagnosis. In Sharon Barner, Ian Harris, Daniel Kroening, and Orna Raz, editors, *Hardware and Software: Verification and Testing: 6th International Haifa Verification Conference, HVC 2010*, pages 29–45. Springer Verlag, 2011.
12. Wenchao Li, Lili Dworkin, and Sanjit A. Seshia. Mining assumptions for synthesis. In Satnam Singh, Barbara Jobstmann, Michael Kishinevsky, and Jens Brandt, editors, *9th IEEE/ACM International Conference on Formal Methods and Models for Codesign, MEMOCODE 2011, Cambridge, UK, 11-13 July, 2011*, pages 43–50. IEEE, 2011.
13. Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 6(1):68–93, January 1984.
14. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 179–190, New York, NY, USA, 1989. ACM.
15. Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57. IEEE Computer Society, 1977.
16. Roni Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Sceince, Rehovot, Israel, 1992.
17. Moshe Y. Vardi. Alternating automata and program verification. In *In Computer Science Today. LNCS 1000*, pages 471–485. Springer-Verlag, 1995.