

Knowledge Compilation for Quantification in Alternating Automata

S. Akshay¹, Alfredo Cantarella², Supratik Chakraborty¹,
Bernd Finkbeiner^{2,3}, Niklas Metzger²

¹Indian Institute of Technology Bombay, Mumbai, India

²CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

³Technical University of Munich, Munich, Germany

{akshayss, supratik}@cse.iitb.ac.in, {first.last}@cispa.de, finkbeiner@cispa.de

Abstract

We present a knowledge compilation approach for existential and universal quantification in alternating automata. Knowledge compilation transforms models into normal forms with special properties that enable efficient answering of questions of interest. For Boolean formulas, several normal forms that have proven effective for existential/universal quantification, and even for functional synthesis, have been studied in the literature. For infinite word automata, quantification is a fundamental operation in verification tasks such as QPTL satisfiability checking and HyperLTL model checking. Existing algorithms rely on nondeterministic infinite word automata, where existential projection can be efficiently performed state-wise, but universal projection requires complementation. Complementing nondeterministic infinite word automata, however, is expensive, making existing algorithms infeasible for automata in practice. Towards addressing this problem, we propose novel knowledge compilation techniques for existential and universal quantification on alternating safety automata. Our approach compiles alternating automata into normal forms where projection can be applied uniformly and efficiently to each state's transition function. Using the compilations for each type of quantification, we can effectively eliminate a sequence of alternating quantifiers in formulas without complementation. Our BDD-based prototype demonstrates the practical effectiveness of our algorithms on a suite of QPTL satisfiability benchmarks.

1 Introduction

Linear Temporal Logic (LTL) has served as a central specification formalism for reasoning about programs since its introduction half a century ago (Pnueli 1977). Dominant techniques for reasoning about LTL typically proceed by translating LTL formulas into automata over infinite words – most notably nondeterministic Büchi automata (Vardi and Wolper 1986; Vardi 1995; Sistla, Vardi, and Wolper 1985) and alternating Büchi automata (Miyano and Hayashi 1984; Vardi 1997) – and then applying automata-theoretic reasoning. Many of these approaches have been realized in state-of-the-art tools (Blahoudek, Duret-Lutz, and Strejček 2020; Duret-Lutz et al. 2022; Meyer, Sickert, and Luttenberger 2018; Meyer and Sickert 2021; Renkin et al. 2022) that routinely scale to large benchmarks, as evidenced in competitions like SyntComp (Jacobs et al. 2022). However, for other applications such as hyperproperty verification,

LTL is not expressive enough, and it becomes necessary to add quantifiers over propositions or even traces. This need was recognized early with the definition of Quantified Propositional Temporal Logic (QPTL) in (Sistla 1983; Sistla, Vardi, and Wolper 1985), which extends LTL with existential and universal quantifiers over propositions. While complexity-theoretic studies of QPTL were done in early work (Sistla 1983; Sistla, Vardi, and Wolper 1985), the development of practically efficient algorithms for QPTL satisfiability and model checking has not received much attention over the years.

The design of practical algorithms for reasoning about expressive logics like QPTL is often a balancing act between representational succinctness of the problem instance, and algorithmic tractability. Consider the problem of existentially quantifying an atomic proposition p from an LTL formula φ . If we convert φ to a Nondeterministic Büchi Automaton (NBA), existentially quantifying p is easy: simply project the p -labels from all state transitions. However, this comes at the cost of representation size, since translating LTL formulas to NBA often incurs an exponential blowup in state space. For universally quantifying p , we must also complement the NBA for $\neg\varphi$ after projecting p -labels from its transitions. Unfortunately, complementing NBA is a computationally expensive operation (Sistla, Vardi, and Wolper 1985; Tsai et al. 2011), even for state-of-the-art tools (Duret-Lutz et al. 2022). Alternating Büchi Automata (ABA) offer a compelling alternative that addresses the representation challenge. ABA generalize nondeterminism by adding universality, allowing the automaton to switch modes: While reading a single input word, it can nondeterministically choose a successor, or it can force the run into multiple parallel paths that must all accept the remainder of the input. ABA are exponentially more succinct than NBA, and support linear-time complementation (Miyano and Hayashi 1984). However, existential quantification is much harder in ABA, since the run is a tree and distinct states reading the same input may impose contradictory requirements on a proposition. For example, for the LTL formula φ represented by the ABA in Figure 1a, we cannot represent $\exists a.\varphi$ by simply removing a -leaves from the ABA.

Algorithms for QPTL satisfiability and model checking that start from ABA representations of LTL formulas typically proceed by converting the ABA into an NBA, thereby

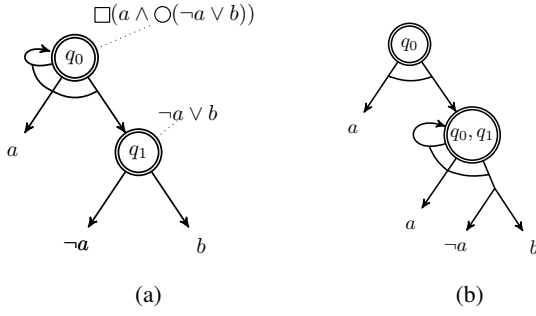


Figure 1: (a) An alternating automaton for the LTL formula $\varphi = \Box(a \wedge \bigcirc(\neg a \vee b))$. (b) The compiled automaton for the existential quantification of the QPTL formula $\exists a.\varphi$. States q_0 and q_1 are in conflict and merged whenever they are reached conjunctively.

reintroducing the exponential state explosion and sacrificing the very succinctness that makes ABA attractive in the first place. More importantly, handling QPTL formulas with quantifier alternations forces one to complement the automaton at each alternation depth, leading to a nonelementary blow-up and rendering such approaches infeasible on complex formulas. This motivates us to ask whether one can represent ABA in a form that supports quantification directly and efficiently, while remaining within the alternating setting, and avoiding the intermediate nonelementary explosion. While the final emptiness check for ABA remains computationally demanding, eliminating repeated expensive complementation shifts the complexity away from this limiting step, making previously intractable formulas accessible.

In this paper, we propose a solution for the above challenge via knowledge compilation. Our work is inspired by the use of normal forms for existential quantification in the setting of Boolean models. Such forms include circuits in decomposable negation normal form (DNNF) (Darwiche 2001a; Darwiche 2001b), weak-DNNF (Akshay et al. 2018; Illner and Kucera 2024), synthesis negation normal form (SynNNF) and variants (Shah et al. 2021; Akshay, Chakraborty, and Shah 2024), for which practically efficient knowledge compilation algorithms have been studied. Knowledge compilation for interval automata, which are generalizations of Binary Decision Diagrams (Bryant 1986), has been studied in (Niveau et al. 2010); however, interval automata are unrelated to automata over infinite words.

Our main novelty lies in extending the knowledge compilation framework for Boolean models to the domain of infinite word automata. Rather than performing a full ABA to NBA translation, we propose a technique to detect and repair only the specific states that cause problems or *conflicts* during quantification. Our approach compiles an alternating automaton into a conflict-free representation where existential projection can be applied *state-wise* to the transition function, preserving the compact alternating structure. Consider again the example in Figure 1a, where one possible ABA for the LTL formula $\varphi = \Box(a \wedge \bigcirc(\neg a \vee b))$ is shown. The compiled automaton for the QPTL formula $\exists a.\varphi$ is shown in Figure 1b. Significantly, the compiled automaton is an ABA language-equivalent to the given ABA, and allows ex-

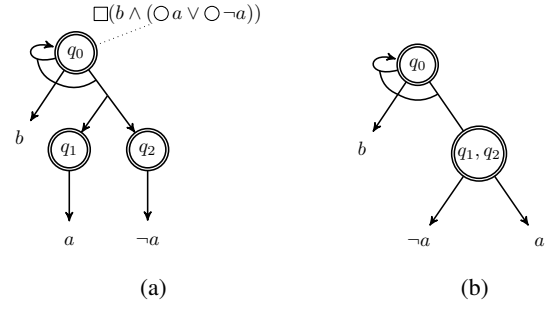


Figure 2: (a) An alternating automaton for the LTL formula $\varphi = \Box(b \wedge (\bigcirc a \vee \bigcirc \neg a))$. (b) The compiled automaton for the universal quantification of the QPTL formula $\forall a.\varphi$. States q_1 and q_2 are in conflict.

istentially quantifying a by simply quantifying a from each state's transition function. Similarly, in Figure 2a and 2b, we show an ABA and its language-equivalent compiled form for universally quantifying a . The ABA corresponding to the quantified formula is obtained by universally quantifying a from each state's transition function in Figure 2b.

We summarize our primary technical contributions below.

- We define knowledge representations for existential and universal quantification for ABA, and develop syntactic conditions for these forms.
- For alternating safety automata (ASA), we identify the notion of *conflicts* for existential quantification, and provide a fixpoint-based compilation algorithm that identifies the set of conflicting states. We also present a repair algorithm based on a *modified Miyano-Hayashi construction* (Miyano and Hayashi 1984) that resolves these conflicts locally.
- We extend our conflict detection and repair mechanisms to support *universal quantification*, enabling practically efficient satisfiability checking for safety QPTL formulas with quantifier alternations.
- We implement our approach in a BDD-based prototype and evaluate it on *QPTL satisfiability* benchmarks against two state-of-the-art QPTL solvers. We demonstrate significant performance improvements by avoiding complementation operations via knowledge compilation.

2 Preliminaries

We consider a finite set of Boolean variables Σ . An infinite trace $t \subseteq (2^\Sigma)^\omega$ is an infinite sequence of subsets of Boolean variables. We refer to the i 'th element of the trace by $t[i]$. An (acyclic) tree $\mathcal{T}$ over a set of directions D is a prefix-closed subset of D^* . The empty sequence ϵ is the root, and the children of a node $n \in \mathcal{T}$ are the nodes $children(n) = \{n \cdot d \in \mathcal{T} \mid d \in D\}$. A Σ -labeled tree is a pair $(\mathcal{T}, r)$, where $r : \mathcal{T} \rightarrow \Sigma$ is the labeling function. A directed acyclic graph (DAG) (V, E) consists of a set of vertices V and a set of edges $E \subseteq V \times V$. We define the set of successors of a vertex v in a DAG (V, E) as $post(v) = \{v' \in V \mid (v, v') \in E\}$. The set of reachable vertices for a vertex v

then is $reach(v) = \{v' \in V \mid \exists v_0 v_1 \dots v_n \text{ s.t. } (v_i, v_{i+1}) \in E, v_0 = v, \text{ and } v_n = v'\}$. We assume DAGs to be acyclic.

Boolean formulas. A Boolean formula over a set of Boolean variables Σ is a combination of the standard operators $\wedge$ (and), $\vee$ (or), $\neg$ (negation), and the derived operators $\rightarrow$ (implication), and $\leftrightarrow$ (equivalence). The semantics of a Boolean formula $B(\Sigma)$ over a set of Boolean variables Σ is a mapping $B : 2^\Sigma \rightarrow \{0, 1\}$. A subset $X \subseteq 2^\Sigma$ is a satisfying assignment of B if $B(X)$ evaluates to 1. We denote the set of all satisfying assignments of B as $Sat(B)$. The set of minimal satisfying assignments is $mSat(B) = \{X \in Sat(B) \mid \forall X' \subset X. X' \notin Sat(B)\}$. We furthermore use $X \models B$ if $X \in mSat(B)$ and $X \not\models B$ if $X \notin mSat(B)$. Setting a variable $a \in \Sigma$ in a Boolean formula B to either *true* or *false* is denoted by $B[a \mapsto 1]$ and $B[a \mapsto 0]$. Setting a subset of variables $A \subset \Sigma$ in B is denoted by $B[A \mapsto 1]$ and $B[A \mapsto 0]$. Existential quantification of a variable $a \in \Sigma$ in a Boolean formula is the Boolean formula $\exists a.B = B[a \mapsto 1] \vee B[a \mapsto 0]$. Universal quantification of a variable $a \in \Sigma$ in a Boolean formula is the Boolean formula $\forall a.B = B[a \mapsto 1] \wedge B[a \mapsto 0]$. We denote the set of *positive* Boolean formulas over a set of variables Σ as $\mathbb{B}^+(\Sigma)$, whereby a Boolean formula is positive if $\neg$ does not occur in its representation.

Alternating Automata. The set of literals $L_\Sigma = \{\neg a \mid a \in \Sigma\} \cup \Sigma$ for a set of Boolean variables contains all positive and negative literals of Σ . We use elements of L_Σ as shorthand for truth assignments to variables in Boolean formulas, and use all notations accordingly. An *alternating Büchi automaton* $\mathcal{A}$ over Boolean variables Σ is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, where: Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \rightarrow \mathbb{B}^+(Q \cup L_\Sigma)$ is the *transition function*, and $F \subseteq Q$ is the Büchi accepting condition. We define the set of direct successors for a state q as $post(q) = \{X \cap Q \mid X \in mSat(\delta(q))\}$. Similarly, we define the set of literals appearing in the transition function of a state q as $l(q) = \{a \in L_\Sigma \mid \exists X \text{ s.t. } X \models \delta(q) \text{ and } a \in X\}$. The set of all reachable states for a state q is defined as $reach(q) = \{q' \in Q \mid \exists q_0 q_1 \dots q_n \text{ s.t. } q_0 = q, q_n = q' \text{ and } q_{i+1} \in \bigcup post(q_i)\}$. A *run* of an alternating automaton on a word $\alpha \in (2^\Sigma)^\omega$ is a $Q \cup L_\Sigma$ -labeled tree $(\mathcal{T}, r)$, where $r(\epsilon) = q_0$ and for all $n \in \mathcal{T}$, if $r(n) = X$, then $\{r(n') \mid n' \in children(n)\}$ satisfies $\delta(q)$, where $q = X|_Q$, and $\alpha[n] = X|_{L_\Sigma}$. A run is *accepting* if for all infinite branches $n_0 n_1 \dots$ of the run tree, the sequence $r(n_0)r(n_1) \dots$ visits states in F infinitely often. A Büchi automaton is *nondeterministic*, if for every state $q \in Q$, the transition function $\delta(q)$, after projection to Q , is a disjunction of states. A Büchi automaton is *universal*, if for every state $q \in Q$, the transition function $\delta(q)$, after projection to Q , is a conjunction of states. An alternating automaton is a *safety* automaton if $F = Q$. For a trace $t \in (2^\Sigma)^\omega$, the *projection* of t onto $\Sigma' \subseteq \Sigma$ is defined as $t|_{\Sigma'} = (t[0] \cap \Sigma')(t[1] \cap \Sigma') \dots$. The *existential quantification* of an alternating automaton $\mathcal{A}$ over Σ for variable $p \in \Sigma$ is the automaton $\exists p.\mathcal{A}$ over alphabet $\Sigma \setminus \{p\}$ such that $\mathcal{L}(\exists p.\mathcal{A}) = \{t|_{\Sigma \setminus \{p\}} \mid \exists t' \in (2^\Sigma)^\omega. t'|_{\Sigma \setminus \{p\}} = t \text{ and } t' \in \mathcal{L}(\mathcal{A})\}$. The *universal quantification* of $\mathcal{A}$ for a variable $p \in \Sigma$ is the automaton $\forall p.\mathcal{A}$ over alphabet $\Sigma \setminus \{p\}$

such that $\mathcal{L}(\forall p.\mathcal{A}) = \{t|_{\Sigma \setminus \{p\}} \mid \forall t' \in (2^\Sigma)^\omega. t'|_{\Sigma \setminus \{p\}} = t \text{ implies } t' \in \mathcal{L}(\mathcal{A})\}$.

QPTL. Quantified Propositional Temporal Logic (QPTL) (Sistla 1983) extends LTL (Pnueli 1977) with quantification over propositional variables. QPTL formulas over a set Σ of Boolean variables are generated by the following grammar, whereby $a \in \Sigma$: $\varphi ::= a \mid \neg \varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \varphi \mathcal{U} \varphi \mid \exists p.\varphi$. The formula a states that the Boolean variable a needs to hold at the current step of a trace, $\bigcirc \varphi$ requires φ to hold in the *next* step and $\varphi_1 \mathcal{U} \varphi_2$ requires φ_1 to hold *until* φ_2 eventually holds. We derive the usual Boolean constants and connectives *true*, *false*, $\vee$, $\rightarrow$, $\leftrightarrow$, as well as the temporal operators *eventually* ($\Diamond \varphi := true \mathcal{U} \varphi$) and *globally* ($\Box \varphi := \neg \Diamond \neg \varphi$). In addition, we use the *universal quantification* ($\forall p.\varphi := \neg \exists p.\neg \varphi$). The LTL part of the formula follows the standard LTL semantics (Pnueli 1977). The existential operator states that there must exist an infinite sequence of assignments of Boolean values to p s.t. the remaining formula holds. We refer to (Sistla 1983) for details. Translating the quantifier-free LTL sub-formula of a QPTL formula written in prenex normal form to an alternating automaton can be done in linear time (Vardi 1997).

3 Normal Forms for Existential and Universal Quantification

The goal of this paper is to construct alternating automata on which existential and universal quantification can be performed state-wise. We begin by defining the quantification of variables on *alternating* automata in a state-wise manner, analogous to how existential quantification is performed on nondeterministic automata via projection. In nondeterministic automata, existential quantification of a variable p can be applied locally to each state's transition function by simply projecting p from the alphabet and existentially quantifying it in the transition formula. This local operation is sound because nondeterministic runs follow a single path through the automaton. We define state-wise existential quantification for alternating automata formally:

Definition 1 (State-wise Existential Quantification). *Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ and let $p \in \Sigma$. The state-wise existential quantification $\exists p.\mathcal{A}$ is the automaton $\mathcal{A}' = (\Sigma', Q, q_0, \delta', F)$, where $\Sigma' = \Sigma \setminus \{p\}$ and $\delta'(q) = \exists p.\delta(q)$.*

State-wise existential quantification applies existential quantification locally to each state's transition function. This operation removes the variable p from the alphabet and quantifies it in each transition formula independently. The universal case is defined analogously, where we apply universal quantification to each state's transition function:

Definition 2 (State-wise Universal Quantification). *Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ and let $p \in \Sigma$. The state-wise universal quantification $\forall p.\mathcal{A}$ is the automaton $\mathcal{A}' = (\Sigma', Q, q_0, \delta', F)$, where $\Sigma' = \Sigma \setminus \{p\}$ and $\delta'(q) = \forall p.\delta(q)$.*

Applying state-wise quantification directly to an arbitrary alternating automaton can dramatically alter its language. To illustrate this, consider our running example from Figure 1a. The automaton represents the LTL formula $\varphi =$

$\Box(a \wedge \Box(\neg a \vee b))$, which requires that a holds at every position, and at the next position either $\neg a$ or b must hold. For the QPTL formula $\exists a.\varphi$, we want to check if there exists an infinite assignment to a such that the formula is satisfied.

If we naively apply state-wise existential quantification to remove a from the automaton, the operation assumes that every evaluation of a was part of the language. This is because the transitions $\delta(q_0) = a \wedge q_1$ and $\delta(q_1) = (\neg a \vee b)$ become $\delta'(q_0) = q_1$ and $\delta'(q_1) = \top$ after existentially quantifying a . This completely ignores the fact that only a being *true* at every position satisfies the original formula while existentially quantifying a in $\delta(q_1)$. This can result in the automaton accepting a superset of the original language, violating language preservation. We now define normal forms for alternating automata where state-wise existential and universal quantifications are language preserving.

Definition 3 (Existential Normal Form). *Let $\mathcal{A}$ be an alternating automaton over Σ . An existential normal form of $\mathcal{A}$ for the Boolean variable $p \in \Sigma$ is an alternating automaton $\mathcal{A}^\exists$ such that $\mathcal{L}(\exists^\circ p.\mathcal{A}^\exists)$ is equivalent to $\mathcal{L}(\exists p.\mathcal{A})$.*

An automaton is in existential normal form for a variable p if applying state-wise existential quantification to p preserves the language of the automaton. We denote the operator $\exists p.\mathcal{A}$ as defined in Section 2. The universal normal form is defined analogously: We require that state-wise *universal* quantification preserves the language of the automaton.

Definition 4 (Universal Normal Form). *Let $\mathcal{A}$ be an alternating automaton over Σ . A universal normal form of $\mathcal{A}$ for the Boolean variable $p \in \Sigma$ is an alternating automaton $\mathcal{A}^\forall$ such that $\mathcal{L}(\forall^\circ p.\mathcal{A}^\forall)$ is equivalent to $\mathcal{L}(\forall p.\mathcal{A})$.*

Before continuing with formal requirements for existential and universal normal forms, we highlight an observation for the automata usually used in verification algorithms.

Proposition 1. *Every nondeterministic automaton is in existential normal form. Every universal automaton is in universal normal form. Every deterministic automaton is in both.*

For nondeterministic automata, an accepting run on an input word w is a single path through the automaton. State-wise existential quantification preserves the language because each state along this path can independently choose a valuation for p that satisfies its transition constraint. For universal automata, an accepting run on an input word w simultaneously visits all states reachable under the universal transitions. State-wise universal quantification preserves the language because each state must provide p and $\neg p$. For deterministic automata, an accepting run on an input word w is a unique path, combining the cases of nondeterministic and universal automata.

4 Syntactic Conditions for Knowledge Representation

In this section, we establish formal conditions that characterize when an alternating automaton is in existential or universal normal form. These conditions form the foundation for the knowledge compilation algorithms presented in Section 5. Rather than operating on the automaton's individual

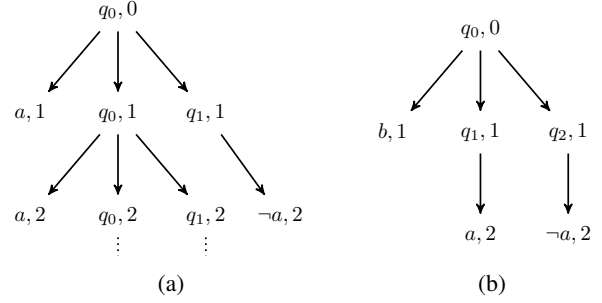


Figure 3: We sketch an existential (a) and universal unfolding (b) of the automata in Figure 1a and Figure 2a, respectively.

runs, our conditions are defined on *partial unfoldings* of the automaton. Partial unfoldings follow the transition function while resolving either nondeterministic or universal choices. These partial unfoldings provide a way to reason about potential conflicts in the automaton's structure without fully exploring all possible input words. For both existential and universal quantification, we define corresponding existential and universal unfoldings and demonstrate how they capture sufficient conditions for the respective normal forms.

4.1 Condition for Existential Normal Forms

The syntactic condition for existential normal forms is formalized on a special unfolding of the automaton, called *existential unfolding*. The general idea is to follow the transition function of the automaton, but resolving nondeterministic choices, i.e., whenever the transition function yields more than one choice for sets of future states, the symbolic existential run chooses exactly one such set. Formally, this yields an infinite acyclic graph:

Definition 5 (Existential Unfolding). *An existential unfolding of an alternating Büchi automaton $\mathcal{A}$ is a DAG (V, E) , with $V \subseteq (Q \cup L_\Sigma) \times \mathbb{N}$, $(q_0, 0) \in V$ and*

$$E \subseteq \bigcup_{i \in \mathbb{N}} \left[(Q \times \{i\}) \times (Q \times \{i+1\}) \right] \cup \left[(Q \times \{i\}) \times (L_\Sigma \times \{i+1\}) \right],$$

and V, E are smallest sets s.t. $\forall (q, i) \in V, \text{post}(q, i) = X$,

- $X \in mSat(\delta(q))$ and $(X \times \{i+1\}) \subseteq V$, and
- $\{(q, i) \times (X \times \{i+1\}) \subseteq E$

The vertices of the DAG are all states and literals for every timestep. We transition from timesteps to timesteps by choosing a set of future states and literals, such that (1) the transition function is satisfied for every state, and (2) the edges point to the correct future vertices. Note that we choose the literals for the variables in Σ for each state, but do not enforce that the variable assignments are unique over all states in every branch of the unfolding. This is necessary to find states that require contradictory valuations for p at the same timestep. Figure 3a sketches an existential unfolding of the automaton in Figure 1a. Every state and literal is duplicated for every timestep, and the edges follow the transition function of the automaton. The unfolding follows all

states of the conjunction in q_0 and chooses one of the disjuncts in q_1 . Using existential unfoldings, we can define the *syntactic condition* for existential quantification.

Definition 6 (Existential Syntactic Condition). *An alternating automaton $\mathcal{A}$ satisfies the existential syntactic condition for $p \in \Sigma$ if for all its existential unfoldings (V, E) and for all $i \in \mathbb{N}$ it holds that either $(p, i) \notin V$ or $(\neg p, i) \notin V$.*

This condition requires that no run contains both p and $\neg p$ at the same timestep. Since all nondeterminism is resolved in the existential unfolding, finding dual literals at the same timestep implies that the automaton traversed through a conjunction of states and arrived in two states where different evaluations for p are required. This happens at timepoint 2 in the existential unfolding in Figure 3a, where the automaton is in both q_1 and q_2 at the same time, which requires $\neg a$ and a , respectively. This is a violation of the syntactic condition, and therefore the automaton in Figure 1a is not in existential normal form for a . This is a critical situation for existential quantification: The state-wise quantification would yield positive results for each state, but the transition functions for both states could be contradictory. In Section 5, we present an algorithm that identifies such conflicts and merges the conflicting states, which results in an automaton that satisfies the syntactic condition.

4.2 Condition for Universal Normal Forms

The universal case follows a dual structure. Rather than resolving nondeterministic choices as in existential unfoldings, universal unfoldings resolve *universal* choices, ensuring that, whenever two states occur at the same timepoint, they are preceeded by nondeterminism. This captures the scenario where different runs of the automaton, arising from nondeterministic choices at earlier points, allow different evaluations of the same variable at the same timestep. We define the universal unfolding accordingly:

Definition 7 (Universal Unfolding). *A universal unfolding of an alternating Büchi automaton $\mathcal{A}$ is a DAG (V, E) , with $V \subseteq (Q \cup L_\Sigma) \times \mathbb{N}$ and $(q_0, 0) \in V$, where*

$$E \subseteq \bigcup_{i \in \mathbb{N}} \left[(Q \times \{i\}) \times (Q \times \{i+1\}) \right] \cup \left[(Q \times \{i\}) \times (L_\Sigma \times \{i+1\}) \right],$$

and V, E are smallest sets s.t. $\forall (q, i) \in V, \text{post}(q, i) = X$,

$$- X \subseteq Q \cup L_\Sigma \text{ is a minimal set s.t. } \forall X' \in m\text{Sat}(\delta(q)),$$

$$X \cap X' \cap Q \neq \emptyset, X' \cap L_\Sigma \subseteq X, \text{ and}$$

$$- (X \times \{i+1\}) \subseteq V \text{ and } \{(q, i)\} \times (X \times \{i+1\}) \subseteq E$$

The universal unfolding follows a dual structure to the existential case: Instead of resolving nondeterministic choices, it resolves *universal* choices in the transition function. For every minimal satisfying assignment X' of $\delta(q)$, i.e., the sets of successor states, at least one state from X' must be included in the chosen set X . Moreover, each literal of each satisfying assignment must be part of the vertices. This ensures that all nondeterministic branches are unfolded and, whenever more than one state appears in the vertices for the same timepoint, these states are reached via disjunctions. The key difference from existential unfoldings

is that universal unfoldings track states that can be reached through *different nondeterministic choices*, capturing scenarios where distinct runs may allow different valuations at the same timestep – this is the situation where state-wise universal quantification would yield *false* spuriously. Figure 3b sketches the universal unfoldings of the automaton in Figure 2a. Using universal unfoldings, we can define the syntactic condition for universal quantification:

Definition 8 (Universal Syntactic Condition). *An alternating automaton $\mathcal{A}$ is in universal normal form for $p \in \Sigma$ if for all its universal unfoldings (V, E) and for all $i \in \mathbb{N}$ it holds that either $(p, i) \notin V$ or $(\neg p, i) \notin V$.*

Figure 3b sketches the universal unfolding of the automaton from Figure 2a, which does not satisfy the syntactic condition. The universal unfolding resolves *universal* choices rather than nondeterministic ones. At each timestep, for each state q , we select a set X of successor states that covers all nondeterministic branches: For every minimal satisfying assignment X' of $\delta(q)$, at least one state from X' must be included in X . In the example, we reach q_1 and q_2 at timepoint 1 and find the conflicting literals a and $\neg a$ at timepoint 2. Note that we do not have $q_0, 1$ in the unfolding since we follow one state of each conjunction and the transition function of q_0 is $(b \wedge q_0 \wedge q_1) \vee (b \wedge q_0 \wedge q_2)$.

We close this section by showing that the syntactic conditions defined in this section imply the normal forms defined in the previous section (precisely, Definition 6 implies Definition 3, and Definition 8 implies Definition 4, respectively).

Theorem 1. *Let $\mathcal{A}$ be an alternating automaton. If $\mathcal{A}$ satisfies the existential syntactic condition, then $\mathcal{A}$ is in existential normal form. If $\mathcal{A}$ satisfies the universal syntactic condition, then $\mathcal{A}$ is in universal normal form.*

Proof. We prove the existential case; the universal case is analogous. Suppose $\mathcal{A}$ is an alternating automaton on input alphabet Σ that satisfies the condition of Definition 6. To establish Definition 3, we must show that $\mathcal{L}(\exists^\circ p. \mathcal{A})$ and $\mathcal{L}(\exists p. \mathcal{A})$ coincide. Recall that $\exists^\circ p. \mathcal{A}$ is simply the automaton $\mathcal{A}$ in which the transition function of every state q has been modified from $\delta(q)$ to $\exists p. \delta(q)$. It follows that for any alternating automaton $\mathcal{A}$, we always have $\mathcal{L}(\exists p. \mathcal{A}) \subseteq \mathcal{L}(\exists^\circ p. \mathcal{A})$: For every word $w \in (\Sigma \setminus p)^\omega$ that belongs to $\mathcal{L}(\exists p. \mathcal{A})$, there exists a sequence of assignments to p , say a_p , corresponding to an accepting run of $\mathcal{A}$ on $w \sqcup a_p$, i.e., w augmented with a_p . The j^{th} element of this sequence, namely $a_p[j]$, serves as a witness assignment to p in $\exists p. \delta(q)$ for every state q of $\mathcal{A}$ visited in the j^{th} step in the same accepting run of $w \sqcup a_p$ on $\mathcal{A}$. Hence, the word w is in $\mathcal{L}(\exists^\circ p. \mathcal{A})$. To complete the proof, it remains to show that $\mathcal{L}(\exists^\circ p. \mathcal{A}) \subseteq \mathcal{L}(\exists p. \mathcal{A})$. This is where Definition 6 is used.

Let $w \in (\Sigma \setminus p)^\omega$ be a word in $\mathcal{L}(\exists^\circ p. \mathcal{A})$. Consider an accepting run of the automaton $\exists^\circ p. \mathcal{A}$ on w . At every step j of this run, the transition formula $\exists p. \delta(q)$ is satisfied for every state q that appears at the j^{th} step of the run. In general, the witness for p used to satisfy $\exists p. \delta(q)$ at step j may vary with the state q . However, since $\mathcal{A}$ satisfies the condition of Definition 6, we know that in every existential unfolding of $\mathcal{A}$, there is no level j at which both p and $\neg p$ appear.

This allows us to construct a sequence a_p of Boolean assignments to p , such that $a_p[j]$ serves as the common witness for p in $\exists p.\delta(q)$ for every state q appearing at the j^{th} step of the above accepting run of $\exists^\circ p.\mathcal{A}$. Specifically, if only p appears in the j^{th} step of the existential unfolding corresponding to the accepting run, we set $a_p[j] = 1$; otherwise, we set $a_p[j] = 0$. Since the same value $a_p[j]$ serves as the witness for $p \in \exists p.\delta(q)$ for all states q appearing at the j^{th} step of the run, it follows that the sequence a_p is a witness sequence of Boolean assignments to p in an accepting run of $\mathcal{A}$ on $w \sqcup a_p$. This implies w is in $\mathcal{L}(\exists p.\mathcal{A})$. $\square$

The syntactic conditions are sufficient for the automaton to be in normal form. In the following section, we develop algorithms for identifying violations of the syntactic conditions and how to repair them. The presented algorithms allow us to transform automata into normal forms for efficient quantification.

5 Knowledge Compilation Algorithms for Safety Automata

In the following, we focus on alternating safety automata. While the definitions in Section 3 apply to all acceptance conditions, the following algorithms operate on the transition structure of the automata, not the acceptance condition, making safety the most suitable choice. The main idea for computing existential and universal knowledge compilations is twofold: We first identify a set of states that need to be conjunctively or disjunctively merged in the automaton, and then perform a subset construction of the automaton for the identified states. We introduce *conflict states* for the first part and present fixpoint algorithms over the transition function of an automaton to iteratively collect conflicting states in the automaton. The second part is two modified versions of the Miyano-Hayashi construction (Miyano and Hayashi 1984), a construction that translates alternating to nondeterministic automata, for only a subset of states and the safety acceptance condition.

5.1 Existential Knowledge Compilation

We begin with the characterization of two states that build an existentially conflicting in an automaton.

Definition 9 (Existential Conflict). *Two states $q, q' \in Q$ are existentially in conflict for p if there exists an existential unfolding (V, E) of $\mathcal{A}$ and $i \in \mathbb{N}$ s.t. (q, i) and $(q', i) \in V$ and there exists an $i' \in \mathbb{N}$ s.t. $(p, i') \in \text{reach}(q, i)$ and $(\neg p, i') \in \text{reach}(q', i)$.*

The states are in existential conflict if they can be reached at the same timepoint in an existential unfolding, i.e., they are below a conjunction, and there is a future timepoint on which p holds on the path from q and $\neg p$ holds on a path from q' . In the example in Figure 1a, q_0 and q_1 are in existential conflict: There exists an existential unfolding where $(q_0, 1)$ and $(q_1, 1)$ are reached, and $(a, 1)$ is reachable from $(q_0, 1)$ and $(\neg a, 1)$ is reachable from $(q_1, 1)$. We show that we can compute whether states are in existential conflict.

Theorem 2. *Deciding whether two states of an alternating automaton $\mathcal{A}$ are existentially in conflict is quadratic in the number of states of $\mathcal{A}$.*

The proof of this theorem can be found in the full version (Akshay et al. 2026) of this paper. Note that Definition 9 is only a sufficient condition based on the syntactic structure of the automaton. We also consider states to be in conflict that are semantically unreachable, i.e., the existential unfolding does not represent a run of the automaton. However, the condition enables us to effectively compute an over-approximation of the states that are in conflict.

Theorem 3. *Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ be an alternating safety automaton and $p \in \Sigma$ a Boolean variable. If $\mathcal{A}$ has no states that are existentially in conflict for p , then $\mathcal{A}$ is in existential normal form for p .*

Proof. We prove that $\mathcal{L}(\exists p.\mathcal{A}) = \mathcal{L}(\exists^\circ p.\mathcal{A})$. $\mathcal{L}(\exists p.\mathcal{A}) \subseteq \mathcal{L}(\exists^\circ p.\mathcal{A})$: Let $w \in \mathcal{L}(\exists p.\mathcal{A})$. Then there exists a $w' \in (2^\Sigma)^\omega$ with $w'|_{\Sigma \setminus \{p\}} = w$ and $w' \in \mathcal{L}(\mathcal{A})$. Let $(\mathcal{T}, r)$ be the accepting run tree of $\mathcal{A}$ on w' . For every node $n \in \mathcal{T}$, the set S of children of n satisfies $\delta(r(n))$. Recall that the state-wise existential quantification $\exists^\circ p.\mathcal{A}$ shares the same set of states as $\mathcal{A}$, but replaces the transition function with $\delta'(r(n)) = \exists p.\delta(r(n))$. Since $\delta(r(n)) \rightarrow \exists p.\delta(r(n))$ is a tautology, and S satisfies $\delta(r(n))$, S also satisfies $\delta'(r(n))$. Thus, $(\mathcal{T}, r)$ is an accepting run tree of $\exists^\circ p.\mathcal{A}$ on w . $\mathcal{L}(\exists^\circ p.\mathcal{A}) \subseteq \mathcal{L}(\exists p.\mathcal{A})$: We prove this by contradiction. Assume $\mathcal{L}(\exists^\circ p.\mathcal{A}) \not\subseteq \mathcal{L}(\exists p.\mathcal{A})$. Thus, there exists a word $w \in \mathcal{L}(\exists^\circ p.\mathcal{A}) \setminus \mathcal{L}(\exists p.\mathcal{A})$. Since $w \in \mathcal{L}(\exists^\circ p.\mathcal{A})$, there exists an accepting run tree $(\mathcal{T}, r)$ of $\exists^\circ p.\mathcal{A}$ on w . Also, since $w \notin \mathcal{L}(\exists p.\mathcal{A})$, and $\exists^\circ p.$ only changes the transition function regarding to p , the transition function of one state of one of the branches of $(\mathcal{T}, r)$ evaluated to *false*, hence rejecting the word w . Specifically, there exists a timestep $i \in \mathbb{N}$ where $(\mathcal{T}, r)$ branches conjunctively to states q_1, q_2 , s.t. for some timestep $i' \geq i$, the branch from q_1 requires p at i' and the branch from q_2 requires $\neg p$ at i' . We then can construct the existential unfolding (V, E) of $\mathcal{A}$ that follows the run tree $(\mathcal{T}, r)$ by guessing the correct nondeterministic choices in the transition function and following the universal branches. Consequently, this existential unfolding contains both (q_1, i) and (q_2, i) and, therefore, (p, i') and $(\neg p, i')$. This is a contradiction to $\mathcal{A}$ being in existential normal form. $\square$

Computing Existential Conflicts. We define a fixpoint algorithm (see Alg. 1) over the alternating automaton that computes a set of states M such that the states in $Q \setminus M$ are not in conflict. The algorithm maintains a labeling function l that maps each state and timestep to a set of sets of literals. Initially, for each state q , we set $l[q, 0]$ to contain only the literals appearing in q 's transition function $\delta(q)$. In each iteration i , we compute $l[q, i]$ by considering all minimal satisfying assignments X of $\delta(q)$ (obtained via $\text{post}(q)$), and for each successor state $q' \in X$, we take the union of all label sets $u \in l[q', i - 1]$ from the previous iteration. This propagates the reachable literals backward through the automaton's transition structure. After updating all labels, we

Algorithm 1: Existential Fixpoint Algorithm

```

1 let existentialConfs( $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ ,  $p \in \Sigma$ ) :=
2   let confs =  $\emptyset$ 
3   for  $q \in Q$  do  $l[q, 0] = \{l(q) \cap L_{\{p\}}\}$ 
4   for  $i = 1; i++$  do
5     for  $q \in Q$  do
6        $l[q, i] = \bigcup_{X \in \text{post}(q)} \{\bigcup_{q' \in X} \bigcup_{u \in l[q', i-1]} u\}$ 
7     for  $q \in Q$  do
8       if  $\exists X \in \text{post}(q). \exists q', q'' \in X.$ 
9          $l[q', i-1] \neq l[q'', i-1]$  do
10          $\text{confs} = \text{confs} \cup \text{reach}(q') \cup \text{reach}(q'') \cup \{q', q''\}$ 
11   if  $\exists i' < i. \forall q' \in Q. l[q', i] = l[q', i']$  return confs

```

check for conflicts: If there exists a state q with two successors q', q'' in some minimal satisfying assignment X such that $l[q', i-1] \neq l[q'', i-1]$, then these successors are reached conjunctively but have different reachable literals at the same timepoint, indicating a conflict. We add both q' and q'' , and all their successors, to the conflict set. The algorithm terminates when a fixpoint is reached, i.e., when there exists a previous iteration i' such that $l[q', i] = l[q', i']$ for all states q' . This occurs after all loop combinations in the automaton have been processed.

We demonstrate the core ideas of the algorithm on our running example in Figure 1a with $p = a$. We initialize the labeling function l with the literals of $L_{\{a\}}$ appearing in each state's transition function: $l[q_0, 0] = \{\{a\}\}$ and $l[q_1, 0] = \{\{\neg a\}\}$. The algorithm then enters the main loop (line 5) for $i = 1$. Since $\text{post}(q_0) = \{\{q_0, q_1\}\}$, it sets $l[q_0, 1] = \{l[q_0, 0] \cup l[q_1, 0]\} = \{\{\neg a, a\}\}$. The state q_1 has no successor states, so $l[q_1, 1] = \emptyset$. Proceeding to the conflict detection (line 8), the algorithm checks if any two states in the conjunction $X = \{q_0, q_1\}$ have differing labels in the previous step ($i = 0$). Since $l[q_0, 0] = \{\{a\}\} \neq \{\{\neg a\}\} = l[q_1, 0]$, the states q_0 and q_1 are added to the conflict set. In the next iteration of the loop ($i = 2$), the labeling remains the same, i.e., $l[q_0, 2] = \{\{\neg a, a\}\}$ and $l[q_1, 2] = \emptyset$. Both states would once again be recognized as conflicts for the previous iteration ($i = 1$). The termination condition (line 11) evaluates to *true* since $l[q_0, 1] = l[q_0, 2]$ and $l[q_1, 1] = l[q_1, 2]$, returning the conflict set $\{q_0, q_1\}$.

Existential Compilation of Alternating Safety Automata.

We adapt the Miyano-Hayashi construction (Miyano and Hayashi 1984) for knowledge compilation for a set of conflicting states. We use a subset construction to repair the conflicts in a set of conflicting states M . The given set M is such that for no $q, q' \in Q \setminus M$ it holds that q is in existential conflict with q' . We convert the transition function of each state to a conjunction of conflict states. For every conjunction of states, we then build the conjunction of their transition functions and proceed to the resulting successor states. Most importantly, whenever a state is reached that is not in M , we preserve the original transition function. This drastically limits the state-explosion induced by the subset construction.

Theorem 4. *Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ be an alternating safety automaton, and let $M \subseteq Q$ be an over-approximation of the existential conflicts in Q for $p \in \Sigma$. There exists an alternating automaton of exponential size in M that is in existential normal form for p .*

Proof. We construct $\mathcal{A}' = (\Sigma', Q', q'_0, \delta', F')$ as follows, where $\Sigma' = \Sigma$ and $F = Q'$.

$$Q' = (2^M) \cup (Q \setminus M), \quad q'_0 = \begin{cases} \{q_0\} & \text{if } q_0 \in M \\ q_0 & \text{else} \end{cases}$$

$$\delta'(q) = \bigvee_{X' \subseteq M} X' \wedge \delta(q)[X' \rightarrow 1, M \setminus X' \mapsto 0]$$

$$\delta'(X) = \bigvee_{X' \subseteq M} X' \wedge \bigwedge_{q \in X} \delta(q)[X' \mapsto 1, M \setminus X' \mapsto 0]$$

In this alternating automaton, there exists no accepting run on which two states $q, q' \in M$ appear on different paths at the same timepoint. It is, therefore, in existential normal form. The automaton is exponential in the size of M . $\square$

5.2 Universal Knowledge Compilation

Universal knowledge compilations change the perspective on conflicting states: We must identify pairs of states on which the universal quantification of a variable in a single state of the automaton would lead to a spurious result. Assume there exist two different runs in an alternating automaton and state q is reached in the first and q' in the second. Furthermore, $\delta(q)$ enforces that p holds, but $\delta(q')$ enforces that $\neg p$ holds on the input word. Universal quantification should evaluate to *true* in this case, since for any evaluation of p , there exists a satisfying assignment, one in $\delta(q)$ and one in $\delta(q')$. However, if Definition 2 is applied to this automaton, $\delta(q)$ and $\delta(q')$ will both evaluate to *false*, spuriously rejecting all words that traverse through q and q' . We solve this by, again, combining both states in the knowledge compilation. In contrast to the existential compilation, we consider conflicting states to be in *disjunction*. We begin with the definition of a universal conflict.

Definition 10 (Universal Conflict). *Two states $q, q' \in Q$ are universally in conflict for p if there exists a universal unfolding (V, E) of $\mathcal{A}$ and $i \in \mathbb{N}$ s.t. (q, i) and $(q', i) \in V$ and there exists an $i' \in \mathbb{N}$ s.t. $(p, i') \in \text{reach}(q, i)$ and $(\neg p, i') \in \text{reach}(q', i)$.*

We ensure that states in universal conflict are preceded by a nondeterministic choice in a state of the automaton, which guarantees that there exist two different accepting runs on which the states are reached at the same timestep. Intuitively, this is the case if the automaton contains a nondeterministic choice that leads to different parts of the automaton. Checking whether two states are in universal conflict is quadratic in the number of states.

Theorem 5. *Deciding whether two states of an alternating automaton $\mathcal{A}$ are universally in conflict is quadratic in the number of states of $\mathcal{A}$.*

We refer to the full version (Akshay et al. 2026) for the proofs of the previous and the next theorem.

Theorem 6. Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ be an alternating safety automaton and $p \in \Sigma$ a Boolean variable. If $\mathcal{A}$ has no states that are universally in conflict for p , then $\mathcal{A}$ is in universal normal form for p .

Computing Universal Conflicts. The universal conflict fixpoint algorithm (see Alg. 2) follows a similar structure to the existential case, but with a crucial difference in how conflicts are detected. As before, we maintain a labeling function l that maps each state and timestep to sets of literals, initialized with $l[q, 0] = \{l(q) \cap L_{\{p\}}\}$ for each state q . The key difference appears in line 7: Instead of checking for conflicts within a single minimal satisfying assignment X (which corresponds to conjunctive branching), we check for conflicts *across different* minimal satisfying assignments X and X' of $\delta(q)$. This reflects the nondeterministic nature of universal conflicts: The automaton can choose different paths (different satisfying assignments) that impose contradictory requirements on the valuation of p . Specifically, if there exist distinct minimal satisfying assignments $X, X' \in \text{post}(q)$ with successors $q' \in X$ and $q'' \in X'$ such that $l[q', i-1] \neq l[q'', i-1]$, we identify a universal conflict. This captures the scenario where different runs of the automaton, arising from nondeterministic choices, reach states with incompatible restrictions on the value of p at the same timestep. We add the conflicting states and their successors to the conflict set and terminate when a fixpoint is reached, which occurs exactly after traversing all loop combinations in the automaton.

We demonstrate Alg. 2 on the example in Figure 2a where $p = a$. We initialize the labeling function l with the literals of $L_{\{a\}}$ appearing in each state's transition function: $l[q_0, 0] = \{\{\neg a, a\}\}$, $l[q_1, 0] = \{\{a\}\}$ and $l[q_2, 0] = \{\{\neg a\}\}$. The algorithm then enters the main loop (line 4) for $i = 1$. Since $\text{post}(q_0) = \{\{q_0, q_1\}, \{q_0, q_2\}\}$, it sets $l[q_0, 1] = \{l[q_0, 0] \cup l[q_1, 0] \cup l[q_2, 0]\} = \{\{\neg a, a\}\}$. The states q_1 and q_2 have no successor states, so $l[q_1, 1] = l[q_2, 1] = \emptyset$. Proceeding to the conflict detection (line 7), the algorithm checks if any two states across two disjunctive branches had differing labels in the previous step ($i = 0$). Note that the structure of $\text{post}(q_0)$ tells us precisely which states can be reached conjunctively or disjunctively: Because q_1 and q_2 are in different *inner* sets of $\text{post}(q_0)$, we know they are reached disjunctively. Since $l[q_1, 0] = \{\{a\}\} \neq \{\{\neg a\}\} = l[q_2, 0]$, the states q_1 and q_2 are added to the conflict set. Analogously, q_0 is added to the conflict set. In the next iteration of the loop ($i = 2$), the labeling remains the same, i.e., $l[q_0, 2] = \{\{\neg a, a\}\}$ and $l[q_1, 2] = l[q_2, 2] = \emptyset$. No new conflicts are detected for the previous iteration ($i = 1$). The termination condition (line 11) evaluates to *true* since $l[q_0, 1] = l[q_0, 2]$, $l[q_1, 1] = l[q_1, 2]$ and $l[q_2, 1] = l[q_2, 2]$, returning the conflict set $\{q_0, q_1, q_2\}$. Note that this is a superset of the conflicts shown in Figure 2a.

Universal Compilation of Alternating Safety Automata. We now present an automaton construction that, given a set of conflicting states, produces an alternating automaton

Algorithm 2: Universal Fixpoint Algorithm

```

1 let universalConflicts( $\mathcal{A} = (\Sigma, Q, q_0, \delta, F), p \in \Sigma$ ) :=
2   let confs =  $\emptyset$ 
3   for  $q \in Q$  do  $l[q, 0] = \{l(q) \cap L_{\{p\}}\}$ 
4   for  $i = 1; i++$  do
5     for  $q \in Q$  do
6        $l[q, i] = \{\bigcup_{X \in \text{post}(q)} \bigcup_{q' \in X} \bigcup_{u \in l[q', i-1]} u\}$ 
7     for  $q \in Q$  do
8       if  $\exists X, X' \in \text{post}(q). \exists q' \in X. \exists q'' \in X'.$ 
9          $l[q', i-1] \neq l[q'', i-1]$  do
10        $\text{confs} = \text{confs} \cup \text{reach}(q') \cup \text{reach}(q'') \cup \{q', q''\}$ 
11   if  $\exists i' < i. \forall q' \in Q. l[q', i'] = l[q', i']$  return confs

```

where these states are disjunctively combined whenever they are reached simultaneously.

Theorem 7. Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ be an alternating safety automaton, and let $M \subseteq Q$ be an over-approximation of the universal conflicts in Q for $p \in \Sigma$. There exists an alternating automaton of exponential size in M that is in universal normal form for p .

Proof. We construct $\mathcal{A}' = (\Sigma', Q', q'_0, \delta', F')$ as follows, where $\Sigma' = \Sigma$ and $F = Q'$.

$$Q' = (2^M) \cup (Q \setminus M), \quad q'_0 = \begin{cases} \{q_0\} & \text{if } q_0 \in M \\ q_0 & \text{else} \end{cases}$$

$$\delta'(q) = \bigwedge_{X' \subseteq M} X' \wedge \delta(q)[X' \mapsto 1, M \setminus X' \mapsto 0]$$

$$\delta'(X) = \bigwedge_{X' \subseteq M} X' \wedge \bigvee_{q \in X} \delta(q)[X' \mapsto 1, M \setminus X' \mapsto 0]$$

In this alternating automaton, there are no two accepting runs for the same word on which $q, q' \in M$ appear at the same timepoint. It is, therefore, in universal normal form. The automaton is exponential in the size of M . $\square$

The construction operates as follows: For each non-conflicting state $q \in Q \setminus M$, we replace transitions to conflicting states with transitions to their corresponding subset representation. For each subset $X \subseteq M$ of conflicting states, we construct a new macro-state in Q' whose transition function combines the original transitions disjunctively. The key observation distinguishing this construction from the existential knowledge compilation is the use of *universal* edges (conjunctions) at the top level of $\delta'(q)$ and $\delta'(X)$, in contrast to the nondeterministic edges (disjunctions) used in both the existential compilation and the original Miyano-Hayashi construction (Miyano and Hayashi 1984). This ensures that when multiple conflicting states would be reached simultaneously, they are instead grouped into a single macro-state that disjunctively combines their transition function and the future reachable states, thereby eliminating the universal conflict while preserving the language of the automaton.

Corollary 1 (Knowledge Compilation Framework). Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ be an alternating safety automaton and $p \in \Sigma$ be a Boolean variable.

1. For existential quantification, we first compute the existential conflicts $M_{\exists}$ with Alg. 1, then apply the existential compilation (cf. Theorem 4) to compute $\mathcal{A}_{\exists}$, and finally perform state-wise quantification $\mathcal{A}'_{\exists} = \exists^{\circ} p. \mathcal{A}_{\exists}$. It holds that $\mathcal{L}(\mathcal{A}'_{\exists}) = \mathcal{L}(\exists p. \mathcal{A})$.
2. For universal quantification, we first compute the universal conflicts $M_{\forall}$ with Alg. 2, then apply the universal compilation (cf. Theorem 7) to compute $\mathcal{A}_{\forall}$, and finally perform state-wise quantification $\mathcal{A}'_{\forall} = \forall^{\circ} p. \mathcal{A}_{\forall}$. It holds that $\mathcal{L}(\mathcal{A}'_{\forall}) = \mathcal{L}(\forall p. \mathcal{A})$.

6 Experiments

We implemented our knowledge compilation approach in a tool called COMPILAA (Metzger et al. 2026), written in F# and built on top of several existing libraries and tools. The implementation uses the C# BDD package¹ for efficient manipulation of Boolean formulas and the FsOmegaLib library² for automaton operations. For LTL-to-automaton translation and emptiness checking, we leverage SPOT (Duret-Lutz et al. 2022) as a backend. All experiments are performed on an Apple M3 Max with 48 GB RAM. We set the per-run timeout to 30 seconds to make the experimental evaluation tractable. .

6.1 Implementation

A key design decision in our implementation is a specialized automaton representation that directly reflects the formal definition of alternating automata. In particular, we represent the transition function $\delta(q)$ of each state q as a single BDD that encodes both the constraints on the input alphabet and the successor state assignments. The BDD for $\delta(q)$ is constructed over a variable set that includes both the alphabet variables Σ and Boolean variables representing states in Q . A satisfying assignment of this BDD simultaneously determines which input letter is read and which successor states are activated. Concretely, given an alphabet $\Sigma = \{a, b\}$ and states $Q = \{q_0, q_1, q_2\}$, the BDD for a transition function is built over the variables $\{a, b, q_0, q_1, q_2\}$. From this BDD representation, we can efficiently extract the minimal satisfying assignments $mSat(\delta(q))$ used in our fix-point algorithms. Each minimal satisfying assignment corresponds to a minimal conjunction of literals that satisfies the transition function. This representation is particularly well-suited for state-wise quantification: Given a variable $p \in \Sigma$ to project, we can directly apply the BDD operations $\exists p$ or $\forall p$ to $\delta(q)$ for each state q independently. Our implementation provides functionality for both universal and existential knowledge compilation, allowing us to handle arbitrary QPTL formulas with quantifier alternations.

6.2 Experimental Evaluation

We use QPTL satisfiability as a case study to obtain a preliminary evaluation of the effectiveness of our knowledge compilation approach. Let $\varphi = Q_0 p_0 \dots Q_n p_n. \varphi'$ be a

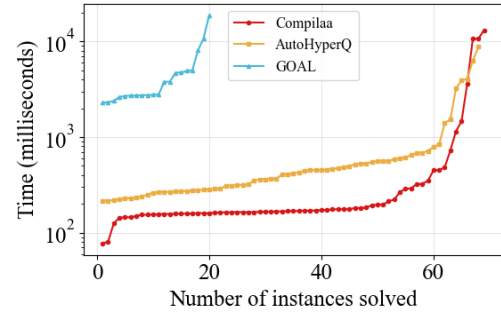


Figure 4: A cactus plot comparing COMPILAA with AUTOHYPERQ and GOAL on SYNTCOMP benchmarks.

QPTL formula where $Q \in \{\exists, \forall\}$ and φ' is the quantifier-free body of the formula. We implemented QPTL satisfiability by applying the results of Corollary 1 from $Q_n p_n$ to $Q_0 p_0$. The starting point is the automaton for φ' , constructed by SPOT. For each Qp , we apply Corollary 1(1) if $Q = \exists$ or Corollary 1(2) if $Q = \forall$. After all $Q_i p_i$ are projected, we check the resulting automaton with SPOT for emptiness. The formula φ is satisfiable if SPOT returns that the automaton is nonempty.

Evaluation. We evaluated our approach on benchmarks derived from the latest edition of the synthesis competition (SYNTCOMP) (Jacobs et al. 2022).³ These benchmarks originate from synthesis problems, which we transform to *universal synthesis* problems expressed as QPTL satisfiability problems. A universal synthesis problem asks whether there are infinite evaluations of output traces for all valuations of environment traces, such that a given temporal specification is satisfied. Note that this is not the realizability problem, where a finite state strategy must be found that produces correct output sequences on the finite past of input traces. We produce QPTL formulas of the form $\forall \vec{i}. \exists \vec{o}. \varphi$, where $\vec{i}$ are the input variables, $\vec{o}$ are the output variables, and φ is an LTL formula expressing the specification. We converted all safety SYNTCOMP benchmarks to this QPTL format and compared our tool COMPILAA to the state-of-the-art QPTL satisfiability solvers AUTOHYPERQ (Beutner and Finkbeiner 2023) and GOAL (Tsai, Tsay, and Hwang 2013). In a second set of benchmarks, we evaluated our tool COMPILAA on QPTL formulas with quantifier alternations per variable. We use SPOT’s RANDLTL to generate 150 random LTL formulas and then convert them to QPTL formulas by assigning quantifiers for each variable in an alternating way. We check the resulting QPTL formulas for satisfiability and compare the execution times of COMPILAA and AUTOHYPERQ. Note that both RANDLTL and the safety specifications of SYNTCOMP limit the scalability of this evaluation. In future work, we imagine that an evaluation on model-checking benchmarks, where COMPILAA is used as a backend, will further highlight the effectiveness of the knowledge compilation algorithms.

¹<https://github.com/microsoft/DecisionDiagrams>

²<https://github.com/ravenbeutner/FsOmegaLib>

³<https://github.com/SYNTCOMP/benchmarks>

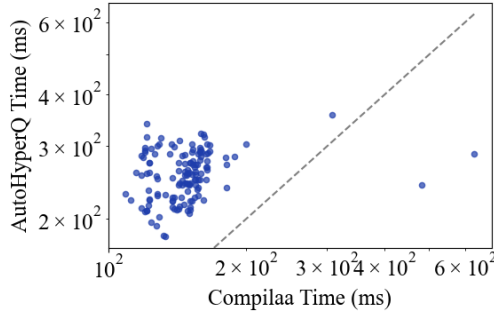


Figure 5: A scatter plot comparing the runtimes of COMPILAA and AUTOHYPERQ on 150 randomly generated QPTL formulas.

Results. Figure 4 presents a cactus plot comparing the performance of COMPILAA against AUTOHYPERQ and GOAL on the SYNTCOMP benchmarks. The x-axis shows the number of solved instances, while the y-axis represents the time taken in seconds. Each point on a curve represents one benchmark instance, sorted by solving time. Our tool COMPILAA performs competitively with state-of-the-art solvers. While GOAL runs into timeouts quickly, COMPILAA and AUTOHYPERQ have comparable runtimes for most benchmarks. However, as problem difficulty increases (moving right), COMPILAA shows improved scalability: It solves almost all instances faster and maintains lower solving times on harder benchmarks. The performance advantage of COMPILAA is clearer with more quantifier alternation depth. Even though the automata for the randomly generated formulas are small, capped by the execution times of RANDLTL, we can observe a clear performance increase of COMPILAA compared to AUTOHYPERQ in Figure 5. This validates our key hypothesis: By avoiding expensive complementation operations through direct knowledge compilation on alternating automata, we can handle quantifications on variables in alternating automata more efficiently. The BDD-based representation and our specialized fixpoint algorithms for conflict detection enable COMPILAA to scale better than approaches that rely on full nondeterminization and repeated complementation steps.

7 Conclusion

We have presented a novel knowledge compilation framework for both existential and universal quantification in alternating safety automata. By compiling automata into existential and universal normal forms, our approach enables efficient state-wise quantification, thereby avoiding the computational bottleneck of repeated complementation used in standard approaches with nondeterministic automata. We provided fixpoint algorithms for detecting quantification conflicts and presented constructions to resolve them locally. The experimental evaluation demonstrates that our tool COMPILAA outperforms existing tools on QPTL satisfiability benchmarks. Future work includes extending the algorithms to Büchi acceptance, where repair for universal quantifiers is hard, and using knowledge compilation as a backend for scalable HyperLTL model checkers.

AI Declaration

The authors have not employed any Generative AI tools.

Acknowledgements

This work originated out of initial discussions in a workshop held as part of the Extended Reunion for Theoretical Foundations of Computer Systems at Simons Institute for the Theory of Computing, Berkeley during July-August 2024. This work was partially supported by the DFG in project 389792660 (TRR 248 – CPEC) and funded by the European Union through ERC Grant HYPER (No. 101055412). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

References

- Akshay, S.; Chakraborty, S.; Goel, S.; Kulal, S.; and Shah, S. 2018. What’s hard about boolean functional synthesis? In Chockler, H., and Weissenbacher, G., eds., *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, 251–269. Springer.
- Akshay, S.; Cantarella, A.; Chakraborty, S.; Finkbeiner, B.; and Metzger, N. 2026. Knowledge compilation for quantification in alternating automata.
- Akshay, S.; Chakraborty, S.; and Shah, S. 2024. Tractable representations for boolean functional synthesis. *Ann. Math. Artif. Intell.* 92(5):1051–1096.
- Beutner, R., and Finkbeiner, B. 2023. Model checking omega-regular hyperproperties with autohyperq. In Piskac, R., and Voronkov, A., eds., *LPAR 2023: Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, 4-9th June 2023*, volume 94 of *EPiC Series in Computing*, 23–35. EasyChair.
- Blahoudek, F.; Duret-Lutz, A.; and Strejček, J. 2020. Semi-nator 2 can complement generalized Büchi automata via improved semi-determinization. In *Proceedings of the 32nd International Conference on Computer-Aided Verification (CAV’20)*, volume 12225 of *Lecture Notes in Computer Science*, 15–27. Springer.
- Bryant. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* C-35(8):677–691.
- Darwiche, A. 2001a. Decomposable negation normal form. *J. ACM* 48(4):608–647.
- Darwiche, A. 2001b. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics* 11(1-2):11–34.
- Duret-Lutz, A.; Renault, E.; Colange, M.; Renkin, F.; Aisse, A. G.; Schlehuber-Caissier, P.; Medioni, T.; Martin, A.;

- Dubois, J.; Gillard, C.; and Lauko, H. 2022. From spot 2.0 to spot 2.10: What's new? In Shoham, S., and Vizel, Y., eds., *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II*, volume 13372 of *Lecture Notes in Computer Science*, 174–187. Springer.
- Illner, P., and Kucera, P. 2024. A compiler for weak decomposable negation normal form. In Wooldridge, M. J.; Dy, J. G.; and Natarajan, S., eds., *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, 10562–10570. AAAI Press.
- Jacobs, S.; Pérez, G. A.; Abraham, R.; Bruyère, V.; Cadilhac, M.; Colange, M.; Delfosse, C.; van Dijk, T.; Duret-Lutz, A.; Faymonville, P.; Finkbeiner, B.; Khalimov, A.; Klein, F.; Luttenberger, M.; Meyer, K. J.; Michaud, T.; Pommellet, A.; Renkin, F.; Schlehuber-Caissier, P.; Sakr, M.; Sickert, S.; Staquet, G.; Tamines, C.; Tentrup, L.; and Walker, A. 2022. The reactive synthesis competition (SYNTCOMP): 2018–2021. *CoRR* abs/2206.00251.
- Metzger, N.; Cantarella, A.; Akshay, S.; Finkbeiner, B.; and Chakraborty, S. 2026. Artifact for the kr 2026 paper "knowledge compilation for quantification in alternating automata".
- Meyer, P. J., and Sickert, S. 2021. Modernising strix. In *10th Workshop on Synthesis*.
- Meyer, P. J.; Sickert, S.; and Luttenberger, M. 2018. Strix: Explicit reactive synthesis strikes back! In *Computer Aided Verification: 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, 578–586. Springer.
- Miyano, S., and Hayashi, T. 1984. Alternating finite automata on omega-words. *Theor. Comput. Sci.* 32:321–330.
- Niveau, A.; Fargier, H.; Pralet, C.; and Verfaillie, G. 2010. Knowledge compilation using interval automata and applications to planning. In Coelho, H.; Studer, R.; and Wooldridge, M. J., eds., *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 459–464. IOS Press.
- Pnueli, A. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, 46–57. IEEE Computer Society.
- Renkin, F.; Schlehuber-Caissier, P.; Duret-Lutz, A.; and Pommellet, A. 2022. Dissecting ltl synt. *Form. Methods Syst. Des.* 61(2–3):248–289.
- Shah, P.; Bansal, A.; Akshay, S.; and Chakraborty, S. 2021. A normal form characterization for efficient boolean skolem function synthesis. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, 1–13. IEEE.
- Sistla, A. P.; Vardi, M. Y.; and Wolper, P. 1985. The complementation problem for büchi automata with applications to temporal logic (extended abstract). In Brauer, W., ed., *Automata, Languages and Programming, 12th Colloquium, Nafplion, Greece, July 15-19, 1985, Proceedings*, volume 194 of *Lecture Notes in Computer Science*, 465–474. Springer.
- Sistla, A. P. 1983. *Theoretical issues in the design and verification of distributed systems*. Ph.D. Dissertation, Harvard University. AAI8403047.
- Tsai, M.-H.; Fogarty, S.; Vardi, M. Y.; and Tsay, Y.-K. 2011. State of büchi complementation. In Domaratzki, M., and Salomaa, K., eds., *Implementation and Application of Automata*, 261–271. Springer Berlin Heidelberg.
- Tsai, M.; Tsay, Y.; and Hwang, Y. 2013. GOAL for games, omega-automata, and logics. In Sharygina, N., and Veith, H., eds., *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, 883–889. Springer.
- Vardi, M. Y., and Wolper, P. 1986. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.* 32(2):183–221.
- Vardi, M. Y. 1995. An automata-theoretic approach to linear temporal logic. In Moller, F., and Birtwistle, G. M., eds., *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, Banff, Canada, August 27 - September 3, 1995, Proceedings)*, volume 1043 of *Lecture Notes in Computer Science*, 238–266. Springer.
- Vardi, M. Y. 1997. Alternating automata: Unifying truth and validity checking for temporal logics. In McCune, W., ed., *Automated Deduction - CADE-14, 14th International Conference on Automated Deduction, Townsville, North Queensland, Australia, July 13-17, 1997, Proceedings*, volume 1249 of *Lecture Notes in Computer Science*, 191–206. Springer.