Language Containment Checking with Nondeterministic BDDs[⋆]

Bernd Finkbeiner

Computer Science Department
Stanford University
Stanford, CA 94305-9045
finkbein@cs.stanford.edu

Abstract. Checking for language containment between nondeterministic ω-automata is a central task in automata-based hierarchical verification. We present a symbolic procedure for language containment checking between two Büchi automata. Our algorithm avoids determinization by intersecting the implementation automaton with the complement of the specification automaton as an alternating automaton. We present a fixpoint algorithm for the emptiness check of alternating automata. The main data structure is a nondeterministic extension of binary decision diagrams that canonically represents sets of Boolean functions.

1 Introduction

Binary decision diagrams (BDDs) have greatly extended the scope of systems that can be verified automatically: instead of searching the entire state space of a model, the verification algorithm works with a symbolic representation of relevant state sets. Symbolic methods have been developed for many verification problems, in particular for temporal logic model checking [CGP99].

For the language containment problem $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ between two ω-automata $\mathcal{A}$ and $\mathcal{B}$, symbolic algorithms have so far only been proposed in the case where $\mathcal{B}$ is deterministic [TBK95]. This is a serious restriction: in property-oriented verification it is advantageous to allow for nondeterminism, since it usually leads to simpler specifications (see [THB95] for examples). Having the same type of automaton for $\mathcal{A}$ and $\mathcal{B}$ also makes hierarchical verification possible, where an intermediate automaton appears as an implementation in one verification problem and as a specification in the next; the verification can follow a chain of increasingly more complex models and ensure that observable properties are preserved.

The standard approach to the language containment check $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ is to first complement $\mathcal{B}$, and then check the intersection with $\mathcal{A}$ for emptiness.

⋆ This research was supported in part by the National Science Foundation grant CCR-99-00984-001, by ARO grant DAAG55-98-1-0471, by ARO/MURI grant DAAH04-96-1-0341, by ARPA/Army contract DABT63-96-C-0096, and by ARPA/AirForce contracts F33615-00-C-1693 and F33615-99-C-3014.

The difficulty with this approach is that the classic constructions for the complementation of ω-automata are all based on determinization. Determinization algorithms for ω-automata, like Safra's construction [Saf88], use an intricate structure to describe deterministic states. Such states not only encode sets of nondeterministic states reachable by the same input prefix, but also keep track of the acceptance status of the nondeterministic computations. Safra-trees have been found to be too complex to be directly encoded in a BDD [THB95].

In our solution we sidestep the determinization construction by intersecting $\mathcal{L}(\mathcal{A})$ and $\overline{\mathcal{L}(\mathcal{B})}$ not in their representation as nondeterministic automata, but in the more general framework of alternating automata, where complementation can be achieved by dualizing the transition function and acceptance condition. This approach makes use of concepts from a new complementation construction by Kupferman and Vardi [KV97]. The use of alternation not only simplifies the algorithm, it also allows us to combine the two automata before any analysis takes place. Thus, no effort is wasted on parts of $\mathcal{B}$ that are not reachable in the combined automaton.

We describe a fixpoint algorithm that checks the resulting alternating automaton for emptiness. This construction involves reasoning about sets of sets of states, one level of aggregation above the sets of states that can be represented by a BDD. We therefore propose an extension to BDDs: by allowing the underlying automaton to be nondeterministic, sets of (deterministic) BDDs can be embedded in a single (nondeterministic) structure.

Overview. In the following Section 2 we briefly survey related work. Section 3 provides background on automata over infinite words. We review deterministic BDDs in Section 4 and present our nondeterministic extension in Section 5. In Section 6 we develop the fixpoint construction for the emptiness check on alternating automata.

2 Related Work

Language containment checking. There are two systems that provide completely automatic language containment checking. Omega [BMUV97] is a package of procedures related to ω-automata and infinite games over finite graphs. Omega implements Safra's construction and uses a completely explicit representation of the state space. HSIS [THB95] is a partially symbolic implementation, again based on Safra's construction. While the state space is still represented explicitly, HSIS makes auxiliary use of BDDs to represent relations on states.

Simulation checking. Simulation is a strictly stronger property than language containment. Tools capable of simulation checking, such as Mocha [AHM$^+$98], can therefore be used to prove language containment (usually with some user interaction), but a failed simulation check does not contradict language containment.

Nondeterministic BDDs. There is a rich literature on extensions to BDDs. In particular the idea to add nondeterminism has been exploited before, but with a different objective: parallel-access diagrams [BD96] interpret

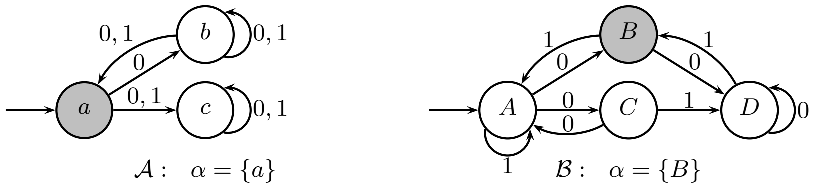
0, 1
0
0, 1
0, 1
0, 1
a
b
c
$\mathcal{A}$: $\alpha = \{a\}$
1
0
1
0
0
0
1
0
1
A
B
C
D
$\mathcal{B}$: $\alpha = \{B\}$

Fig. 1. Büchi automata $\mathcal{A}$ and $\mathcal{B}$. Accepting states are shown in gray.

nondeterminism as disjunction to achieve a more compact representation of certain Boolean functions. Takagi et al. [TNB$^+$97] show that certain methods for the satisfiability checking of combinatorial circuits and techniques that represent Boolean functions as sets of product terms can be regarded as nondeterministic BDDs.

Alternation. Muller and Schupp [MS87] observed that complementing an alternating automaton corresponds to dualizing the transition function and acceptance condition. The application of alternation in verification methods has been studied both for automata-based algorithms [Var95] and in deductive verification [MS00]. Alternating automata have been used in a new complementation construction for Büchi automata [KV97].

3 Automata on Infinite Words

Automata on infinite words differ from automata on finite words in their acceptance mechanism: there are no final states; instead, acceptance is determined w.r.t. the set of states that are visited infinitely often. Different types of acceptance conditions are studied (see [Tho94] for an overview). In the following we will work with Büchi conditions.

Definition 1. *A (nondeterministic)* Büchi-automaton $\mathcal{A} = \langle \Sigma, Q, \theta, \rho, \alpha \rangle$ *consists of a finite input alphabet* Σ*, a finite set of states* Q*, a set of initial states* θ*, a transition function* $\rho : Q \times \Sigma \to 2^Q$ *and a set of accepting states* $\alpha \subseteq Q$*.*

A *run* of $\mathcal{A}$ on an input string $l_0, l_1, \ldots \in \Sigma^\omega$ is an infinite sequence of states $\sigma = v_0, v_1, \ldots$ s.t. $v_0 \in \theta$ and for every $i \geq 0$, $v_{i+1} \in \rho(v_i, l_i)$, i.e., the first state is an initial state and each successor state is included in the successor set given by the transition function.

A run is *accepting* if some accepting state is visited infinitely often. The *language* $\mathcal{L}(\mathcal{A})$ of a Büchi automaton consists of those input strings that have accepting runs.

Example 1. The automaton $\mathcal{A}$ in Figure 1 accepts all infinite words over the alphabet $\{0, 1\}$ that begin with 0 and contain infinitely many 0s. Since $\mathcal{B}$ does not accept the word 0^ω, $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$.

The branching mode in a nondeterministic automaton is existential; a word is accepted if its suffix is accepted in one of the successor states. Alternating

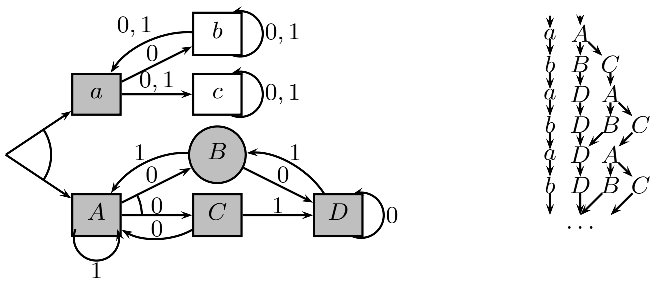

Fig. 2. Alternating automaton $\mathcal{C}$ and a computation for input word 0^ω. Accepting states $\alpha = \{a, A, B, C, D\}$ are shown in gray, stable states $\beta = \{a, b, c, A, C, D\}$ as boxes.

automata combine existential branching with universal branching. Again, many different acceptance conditions are studied. We will work with a combined Büchi and co-Büchi condition.

Definition 2. *An* alternating automaton *is a tuple $\mathcal{A} = \langle \Sigma, Q, \theta, \rho, \alpha, \beta \rangle$ with Σ, Q, α as before; a set of stable states β, a set of initial state sets $\theta \in 2^{2^Q}$; and the transition function $\rho : Q \times \Sigma \rightarrow 2^{2^Q}$, a function from states and alphabet letters to sets of successor state sets.*

A *run* of an alternating automaton is a directed acyclic graph (dag) (N, E), where the nodes are labeled with states $state : N \rightarrow Q$. It is often useful to view the dag as a sequence of sets of nodes which we call *slices*: the i-th slice is the set of nodes that are reached after traversing i edges from root nodes. We call the set of states that occur on the nodes of the i-th slice the i-th configuration. Let configuration 0 be the *root configuration*, and, for finite segments of a run, call the first configuration *source* and the last configuration *target*.

In a run for the input string $l_0, l_1, \ldots \in \Sigma^\omega$, the root configuration is one of the sets in θ, and, for each state v in the i-th configuration, the set of states on successor nodes is one of the successor sets in $\rho(v, l_i)$. A run is accepting if every path visits some α-state infinitely often, and eventually only visits states in β.

Finding a Büchi automaton that accepts the complement of a nondeterministic Büchi automaton is complicated and leads to an exponential blow-up [Saf88]. Alternating automata can be complemented without blow-up by dualizing the transition function and acceptance condition [MS87]. Thus, it is also very simple to construct an alternating automaton that accepts those words that are accepted by the first but not by the second automaton:

Theorem 1. *For two Büchi automata $\mathcal{A}_1 = \langle \Sigma, Q_1, \theta_1, \rho_1, \alpha_1 \rangle$, $\mathcal{A}_2 = \langle \Sigma, Q_2, \theta_2, \rho_2, \alpha_2 \rangle$ (where $\rho_2(p, l) \neq \emptyset$ for all $p \in Q_2, l \in \Sigma$), the alternating automaton $\mathcal{A} = \langle \Sigma, Q, \theta, \rho, \alpha, \beta \rangle$ with*

- $Q = Q_1 \cup Q_2$,
- $\theta = \{\theta_2 \cup \{p\} \mid p \in \theta_1\}$,
- $\rho(s, a) = \mathit{if}\ (s \in Q_1)\ \mathit{then}\ \{\{p\} \mid p \in \rho_1(s, a)\}\ \mathit{else}\ \{\rho_2(s, a)\}$,
- $\alpha = \alpha_1 \cup Q_2$,

– $\beta = (Q_2 \backslash \alpha_2) \cup Q_1$

accepts the language $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \overline{\mathcal{L}(\mathcal{A}_2)}$.

Example 2. An alternating automaton for the language $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(\mathcal{B})}$ is shown in Figure 2.

4 Binary Decision Diagrams

A binary decision diagram (BDD) [Bry86] is a data structure for the representation of Boolean functions $f : \mathbb{B}^n \to \mathbb{B}$. In their reduced and ordered form, BDDs represent Boolean functions canonically for fixed variable orderings. For many examples BDDs significantly outperform other representations. BDDs can be used to store sets of states, represented by their characteristic function: Boolean "or" corresponds to set union, "and" to intersection. BDDs are also used to represent relations on states, such as the transition function of an automaton. This is done by adding a second "primed" copy for each variable.

Definition 3 (BDD). *A (deterministic)* binary decision diagram *(BDD)* $(\mathcal{V}, Q, E_0, E_1, \phi)$ *is a directed acyclic graph with internal nodes* Q*, edges* $E_0 \cup E_1$*, a single root* ϕ *and two terminal nodes* $\mathbf{0}, \mathbf{1}$*. Each internal node* $n \in Q$ *has exactly two departing edges* $low(n) \in E_0, high(n) \in E_1$*. Every internal node* $n \in Q$ *is labeled with a variable* $var(n) \in \mathcal{V}$*.*

The successor nodes along the $low(n)$ and $high(n)$ edges are referred to as the *low* and *high* successors of n. A BDD d with root node ϕ defines a Boolean function $f_d = f_\phi : \mathbb{B}^n \to \mathbb{B}$ as follows:

– the terminal node **1** defines the constant function *true*.
– the terminal node **0** defines the constant function *false*.
– an internal node $n \in Q$ represents the function

$$f \;:\; (\text{if } var(n) \text{ then } f_1 \text{ else } f_0)$$

where f_0, f_1 are the functions represented by the *low* and *high* successors, respectively.

Of special interest are BDDs in a canonical form called reduced and ordered.

Definition 4. *A BDD is* ordered *(OBDD), if on all paths through the graph the labeling respects a given linear order on the variables* $v_1 > v_2 > \cdots > v_n$*; i.e., on all paths through the graph, smaller variables are traversed first. An OBDD is* reduced *(ROBDD) if*

1. *no two different internal nodes have the same label* and *the same high and low successors,*
2. *no internal node has identical high and low successor.*

Theorem 2. *[Bry86] For any Boolean function* $f : \mathbb{B}^n \to \mathbb{B}$ *and a given variable ordering, there is (up to isomorphism) exactly one ROBDD* d *s.t.* $f_d = f$*.*

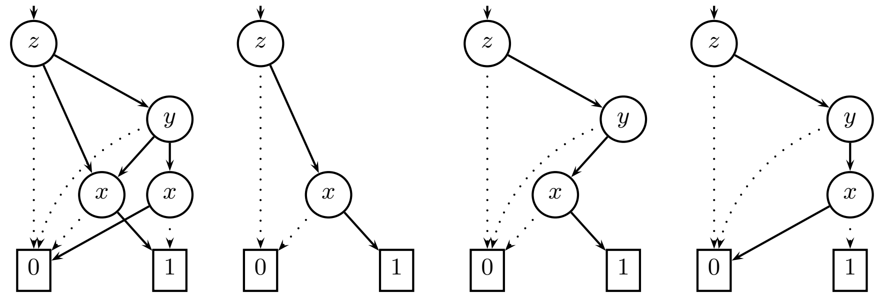

Fig. 3. Nondeterministic BDD (left) and three embedded deterministic BDDs. Solid edges are *high*-successors, dotted edges *low*-successors.

5 Nondeterministic Binary Decision Diagrams

For the analysis of alternating automata we need a more expressive representation than BDDs. Sets of sets of states as they occur, for example, in the initial condition or as sets of configurations, cannot be represented as a conventional BDD. The extension we present in this section uses nondeterministic BDDs to represent sets of Boolean functions. We interpret the nondeterministic BDD to describe the set of all deterministic BDDs that can be embedded in it.

Example 3. Figure 3 shows a nondeterministic BDD and the three embedded deterministic BDDs.

Nondeterministic BDDs may have more than one root node, and the out-degree of internal nodes may be higher than two, so we consider the sets of *High* and *Low* departing edges.

Definition 5. *A nondeterministic binary decision diagram (NBDD) $(\mathcal{V}, Q, E_0, E_1, \Phi)$ is a directed acyclic graph with internal nodes Q, edges $E_0 \cup E_1$, a set of root nodes $\Phi \subseteq Q$, and two terminal nodes $\mathbf{0}, \mathbf{1}$. The set of departing edges from an internal node $n \in Q$ is partitioned into $Low(n) \subseteq E_0$ and $High(n) \subseteq E_1$. Every internal node $n \in Q$ is labeled with a variable $var(n) \in \mathcal{V}$.*

A NBDD D with root set Φ defines a set of Boolean functions $\mathcal{F}_D = \mathcal{F}_\Phi \subseteq 2^{\mathbb{B}^n \to \mathbb{B}}$ as follows:

- the terminal node $\mathbf{1}$ defines the set $\mathcal{F}_\mathbf{1} = \{true\}$.
- the terminal node $\mathbf{0}$ defines the set $\mathcal{F}_\mathbf{0} = \{false\}$.
- a set of nodes Ψ defines the union of the sets represented by the individual nodes: $\mathcal{F}_\Psi = \bigcup_{n \in \Psi} \mathcal{F}_n$.
- for an internal node $n \in Q$, let $\mathcal{H}, \mathcal{L}$ denote the sets defined by its *High* and *Low* successors, respectively. Then n defines the set:

$$\mathcal{F} = \left\{ \begin{array}{r} (\text{if } var(n) \text{ then } f_1 \text{ else } f_0) \\ \text{s.t. } f_0 \in \mathcal{L} \text{ and } f_1 \in \mathcal{H} \end{array} \right\}$$

BDDs are therefore a special (deterministic) case of NBDDs: for a given BDD $(\mathcal{V}, Q, E_0, E_1, \phi)$ the NBDD $(\mathcal{V}, Q, E_0, E_1, \{\phi\})$ characterizes the singleton set containing the Boolean function defined by the BDD.

Definition 6. *A BDD $d = (\mathcal{V}, Q^d, {E_0}^d, E_1^d, \phi^d)$ is* embedded *in an NBDD $D = (\mathcal{V}, Q^D, {E_0}^D, E_1^D, \Phi^D)$ iff there is a simulation function $\gamma : Q^D \to Q^d$ with $\gamma(\mathbf{0}) = \mathbf{0}, \gamma(\mathbf{1}) = \mathbf{1}$, $\phi^d \in \gamma(\Phi^D)$ and for all nodes $n \in Q^D$, $var(n) = var(\gamma(n))$, if $\gamma(n')$ is the low^d-successor of $\gamma(n)$ then n' is a Low^D-successor of n, if $\gamma(n')$ is the $high^d$-successor of $\gamma(n)$ then n' is a $High^D$-successor of n.*

We say that two node sets Φ_1, Φ_2 in an NBDD are *mutually exclusive* iff there is no BDD that is embedded in both the NBDD with root node set Φ_1 and the NBDD with root node set Φ_2. The notions of ordered and reduced diagrams can now be lifted to NBDDs:

Definition 7. *A NBDD is* ordered *(ONBDD), if on all paths through the graph the labeling respects a given linear order on the variables $v_1 > v_2 > \ldots > v_n$. An ONBDD is* reduced *(RONBDD) if*

1. *no two different internal nodes have the same label* and *the same High and Low successor sets,*
2. *the High and Low successor sets of an internal node are mutually exclusive.*

Theorem 3. *Let d be a ROBDD and D a ONBDD with the same variable order. d is embedded in D iff $f_d \in \mathcal{F}_D$.*

Proof. By structural induction. □

RONBDDs are not a canonical representation of sets of Boolean functions. To achieve canonicity, more restrictions on the grouping of functions (if v then f^1 else f^0) that have a common negative cofactor f^0 or a common positive cofactor f^1 are necessary. One such restriction, which we will call the *negative-normal form*, is to require that the functions are grouped by their negative cofactors f_0.

Definition 8. *A RONBDD $D = (\mathcal{V}, Q, E_0, E_1, \Phi)$ is in* negative-normal form *iff the following holds for all nodes $n \in Q$:*

1. *there is only one low-successor: $Low(n) = \{low(n)\}$,*
2. *the low-successor is a BDD,*
3. *no two different High-successors or root nodes are labeled by the same variable* and *have the same low-successor.*

Theorem 4. *For any set of Boolean functions $\mathcal{F} \subseteq 2^{\mathbb{B}^n \to \mathbb{B}}$ and a given variable ordering, there is (up to isomorphism) exactly one RONBDD D in negative-normal form s.t. $\mathcal{F}_D = \mathcal{F}$.*

Proof. We show, by induction on m, that for any subset of the set of variables $\{v_1, \ldots, v_m\} \subseteq \{v_1, \ldots, v_n\}$ (with variable order $v_1 > v_2 > \cdots > v_n$), any set of functions $\mathcal{F}_m \subseteq (2^{\mathbb{B}^n \to \mathbb{B}})$ that only depend on variables in $\{v_1, \ldots, v_m\}$ can be canonically represented by a RONBDD in negative-normal form. In the following we will assume sharing of subgraphs, and identify NBDDs by their root node sets, BDDs by their root node.

$m = 0$: There are four different sets of functions not depending on any variable: $\emptyset, \{\mathit{true}\}, \{\mathit{false}\}, \{\mathit{true}, \mathit{false}\}$. These sets are uniquely represented by the RONBDDs with root node sets $\emptyset, \{\mathbf{0}\}, \{\mathbf{1}\}, \{\mathbf{0}, \mathbf{1}\}$, respectively.

$m \to m+1$: We construct the set of root nodes F for a set $\mathcal{F}_{m+1}$, where v_{m+1} is the least variable some function in $\mathcal{F}_{m+1}$ depends on. For each function f in $\mathcal{F}_{m+1}$ we consider the positive and negative cofactor $f^b(x_1, \ldots, x_{m+1}, \ldots, x_n) = f(x_1, \ldots, b, \ldots, x_n), b \in \mathbb{B}$ [the $(m+1)$st argument is replaced by b]. This allows us to separate the subset of functions A that do not depend on v_{m+1}:

$$A = \{\ f \mid f \in F_{m+1} \text{ and } f^0 = f^1\ \}.$$

For all other functions we separate the positive and negative cofactor in the following set of pairs:

$$B = \{\ (f^0, f^1) \mid f \in F_{m+1} \text{ and } f^0 \neq f^1\ \}.$$

Next, we group the positive cofactors by the negative cofactors:

$$C = \{\ (f, X) \mid \exists g\ .\ (f, g) \in B,\ X = \{g \mid (f, g) \in B\}\ \}.$$

The resulting sets of positive cofactors contain only functions that do not depend on v_{m+1}. The same holds for the set of functions in A. By the induction hypothesis, we can therefore find negative-normal RONBDDs as follows:

$$\begin{aligned} D = \{\ (d_f, D_Y) \mid\ & d_f \text{ is the canonical ROBDD for } f, \\ & D_Y \text{ is the root node set of the canonical RONBDD} \\ & \quad \text{for } Y \text{ with } (f, Y) \in C\ \}, \\ E =\ & \text{the root node set of the canonical RONBDD for } A. \end{aligned}$$

Finally, we can construct the set of root nodes for $\mathcal{F}_{m+1}$:

$$F = \{\langle \mathit{var} = v_{m+1}, \mathit{low} = d_f, \mathit{High} = D_Y \rangle \mid (d_f, D_Y) \in D\} \cup E.$$

The constructed NBDD is ordered, reduced and in negative-normal form since the NBDDs in D and E are, and the newly constructed nodes maintain all conditions. It remains to show that the RONBDD is unique.

Assume there was a different negative-normal RONBDD with root node set F' defining $\mathcal{F}_{m+1}$. Consider the functions in $\mathcal{F}_{m+1}$ that do not depend on v_{m+1}: since the *High* and *Low* successors of any node must be mutually exclusive, they cannot be contained in the set represented by a node labeled by v_{m+1} (reducedness). By the induction hypothesis we know that the set of all nodes in F that are not labeled by v_{m+1} is canonical (the functions represented by the subset depend only on greater variables). Thus F and F' must differ in nodes that are both labeled by v_{m+1}.

Suppose there are two functions f_1, f_2 that are characterized by the same root node in one diagram but by two different root nodes in the other. All functions characterized by the same node in a ROBDD in negative-normal form have the same negative cofactor (conditions 1 and 2 and Theorem 2). Thus the diagram that represents them on two different nodes cannot be in negative-normal form (condition 3). □

```
UNION($N, M$)
1 $R \leftarrow (N \cup M) \cap \{\mathbf{0}, \mathbf{1}\}$
2 for all $n \in N$
3   if $\exists m \in M \;.\; var(n) = var(m), low(n) = low(m)$
4     then $R \leftarrow R \;\cup\; \{\; \langle var(n), low(n), \text{UNION}(High(n), High(m)) \rangle \;\}$
5   else $R \leftarrow R \cup \{n\}$
6 for all $m \in M$
7     if $\nexists n \in N \;.\; var(n) = var(m), low(n) = low(m)$
8     then $R \leftarrow R \cup \{m\}$
9 return $R$
```

Fig. 4. Operation UNION, computing the union of two sets represented by negative-normal NBDDs.

It is straightforward to implement traditional BDD operations (like the application of boolean operations, variable substitution, or quantification) and set operations on NBDDs. As an example, consider UNION, shown in Figure 4. We assume sharing of subgraphs and identify BDDs with their root nodes and NBDDs with their root node sets. The UNION operation computes the negative-normal RONBDD representing the union of two sets represented by two negative-normal RONBDDs. This is done by considering corresponding nodes in the two root node sets. Two nodes *correspond* if they are labeled with the same variable and have the same *low*-successor. The union is computed by recursing on pairs of corresponding nodes and simply adding nodes that do not have a corresponding node in the other set.

6 Emptiness of Alternating Automata

As discussed in Section 3, the language containment problem between nondeterministic Büchi automata is easily reduced to the emptiness problem of alternating automata. In this section we develop a fixpoint algorithm for the emptiness problem. The reachable configurations of an alternating automaton can be computed in a forward propagation from θ. To decide if the finite dag leading to such a configuration can be completed into an accepting run we identify gratifying segments, i.e., segments that would, if repeated infinitely often, form the suffix of an accepting run.

Gratifying segments. Consider an alternating automaton $\mathcal{A} = \langle \Sigma, Q, \theta, \rho, \alpha, \beta \rangle$. A run segment is a finite dag (N, E), where the nodes are labeled with states $state : N \rightarrow Q$, such that for each state v in a configuration, the set of states on successor nodes is one of the successor sets in $\rho(v, l)$ for some input letter $l \in \Sigma$. We characterize gratifying segments w.r.t. a complete preorder $\preceq$ on the states in the source configuration. It will be helpful to identify nodes that are on some path from a source node p to a target node p', s.t. $state(p) \approx state(p')$; we call such nodes *fixed*. A run segment S is *gratifying* if

1. the source and target configuration are the same,
2. all fixed nodes are labeled by β-states,
3. all paths in S visit a node with an α-state,
4. all paths originating from a source node labeled by a state p lead to nodes in the target slice that are labeled with states equivalent to, or smaller than p, and
5. all paths originating from a source node labeled by a state p that visit a non-fixed node lead to target nodes that are labeled with states strictly smaller than p.

Example 4. The segment from slice 2 to slice 4 (configurations $\{a, D, A\}$, $\{b, D, B, C\}$, $\{a, D, A\}$) of the computation in Figure 2 is gratifying w.r.t. the preorder $a \approx D \prec A$. In slices 2 and 4 all nodes are fixed; in slice 3 the nodes labeled by b, D and C are fixed.

Lemma 1. *Let L be a gratifying run segment of an alternating automaton, and P a finite run prefix leading to the source slice of L. Then the dag $G = P \cdot L^\omega$, constructed by appending an infinite number of copies of L to P, is a computation of $\mathcal{A}$.*

Proof. All paths in L visit an accepting state; the paths in L^ω therefore visit an accepting state infinitely often. A path that does not visit a fixed node in L leads to a target node that is labeled by a strictly smaller state than the state on the node it visited in the source slice. Thus, since there are only finitely many states in the source configuration of L, every path in L^ω eventually visits a fixed node. From there, a path can either (1) stay forever in fixed nodes (and therefore in stable states) or (2) visit a non-fixed node and, again, lead to a target node with a strictly smaller state. Hence, eventually (1) must occur. □

Lemma 2. *Let G be a computation of the alternating automaton $\mathcal{A}$. There is a preorder $\preceq$ and a finite prefix P of G that leads to a segment L that is gratifying w.r.t. $\preceq$.*

Proof. For the given computation G we apply a ranking construction by Kupferman and Vardi [KV97]. Consider the following sequence of subgraphs of G.

- $G_0 = G$.
- $G_{2i+1} = G_{2i}$ minus all nodes from which there are only finitely many nodes reachable. Assign rank $2i$ to all the subtracted nodes.
- $G_{2i+2} = G_{2i+1}$ minus all nodes from which only nodes with β-states are reachable. Assign rank $2i + 1$ to all the subtracted nodes.

$G_{2|Q|+1}$ is empty [KV97], i.e., the number of ranks is bounded. There must be infinitely many occurrences of some configuration x, s.t. the nodes with the same state label have the same rank in the two occurrences. We select two occurrences s.t. all paths on the run segment L between them visit an α-state and a node with odd rank. L is a gratifying segment with the order $\preceq$ induced by the ranking. The fixed states have odd rank, non-fixed states even rank. Along a path the rank never increases. □

Annotated configurations. To recognize gratifying segments we keep track of the gratification conditions in configurations. An *annotated configuration* is a tuple $\langle x, f, t, u, \preceq\rangle$ where x is a set of states, t, u are subsets of x, f is a subset of $\beta \cap x$, and $\preceq$ is a complete preorder on x. The goal is to capture the states on fixed nodes in f, "trapped" states (i.e., states on nodes s.t. all originating paths visit a fixed node) in t, and "fulfilling" states (i.e., states on nodes s.t. all paths that originate from this node visit an α-node) in u. We now introduce constraints that ensure that these sets are propagated consistently in a sequence of annotated configurations. Consider two consecutive configurations $\langle x, f, t, u, \preceq\rangle$ and $\langle x', f', t', u', \preceq'\rangle$. We require that there exists a letter l of the input alphabet s.t. for each state $v \in x$ there is a set $y_v \in \rho(v, l)$ so that the following constraints are satisfied:

1. for all $v \in x$, $y_v \subseteq x'$,
2. for all $v' \in f'$ there is a $v \in f$ s.t. $v' \in y_v$,
3. for all $v \in f$, $f' \cap y_v \neq \emptyset$,
4. for all $v \in t - f$, $y_v \subseteq t'$,
5. for all $v \in u - \alpha$, $y_v \subseteq u'$,
6. for all $v' \in f'$ and $v \in x - f$, s.t. $v' \in y_v$, there is a $w \in f$ s.t. $v' \in y_w$ and $w \prec v$,
7. for all $v' \in f'$ and all $w' \in x'$ with $w' \prec' v'$, there exists a $v \in f$ s.t. $v' \in y_v$ and for all $w \in x$ with $w' \in y_w$, $v \prec w$, and
8. for all $v \in f$ s.t. there is a $w \in x$ with $w \prec v$, there exists a $v' \in f'$ with $v' \in y_v$ s.t. for all $w' \in y_w$, $w' \prec' v'$.

Let Y be a set of annotated configurations. We say that an annotated configuration a is *eventually accepting* w.r.t. Y iff there is a sequence of annotated configurations, where a is the first and some $b \in Y$ the last configuration, and where every two consecutive configurations satisfy the constraints above. Let $\text{EventualAccept}(Y)$ denote the set of annotated configurations that are eventually accepting w.r.t. Y.

Lemma 3. *Let S be a gratifying segment leading from a configuration x back to x; then there is an annotation for the source configuration $a = \langle x, f, t = x, u = x, \preceq\rangle$ and an annotation for the target configuration $a' = \langle x' = x, f' = f, t' = f, u' = x \cap \alpha, \preceq\rangle$ s.t. for every set Y of annotated configurations that includes a', $a \in \text{EventualAccept}(Y)$.*

Proof. First, we construct a segment S' in which every path visits a fixed node (by appending as many copies of S as needed). For each slice s in S' we define the following annotated configuration $\langle x_s, f_s, t_s, u_s, \preceq_s\rangle$:

- x_s contains the states on nodes in s,
- f_s contains exactly the states on the fixed nodes in s,
- t_s contains exactly the states on those nodes for which all paths that originate from the node visit a fixed node,
- u_s contains exactly the states on those nodes for which all paths that originate from the node visit an α-node,

– $\preceq_s$ is the following preorder:
  for two states v, w on nodes p, q that are both in f_s or both in $x_s - f_s$, $v \prec_s w$ iff there is a target node q' reachable from q s.t. for all target nodes p' reachable from p, $\mathit{state}(p') \prec \mathit{state}(q')$;
  for two states $v \in f_s, w \in x_s - f_s$ on nodes p, q, $v \prec_s w$ iff there is a target node q' reachable from q s.t. for all target nodes p' reachable from p, $\mathit{state}(p') \preceq \mathit{state}(q')$.

The resulting sequence of annotated configurations satisfies the constraints. Due to space limitations we skip the detailed argument here. □

Let Unmark(X) denote the set of annotated configurations, s.t. $\langle x, f, f, x \cap \alpha, \preceq\rangle \in$ Unmark(X) for $\langle x, f, t, u, \preceq\rangle \in X$. Let Filter($X$) be the subset of the set of annotated configurations X s.t. $u = x, t = x$.

Lemma 4. *Let $a = \langle x, f, t, u, \preceq\rangle$ be an annotated configuration in a set Y s.t. $Y =$ Filter(EventualAccept(Unmark(Y))). Then there is a gratifying segment S.*

Proof. Because of constraint (1) there is a run segment S corresponding to the sequence of configurations in the construction of EventualAccept. We show that S is gratifying. For a slice s, let $\langle x_s, f_s, t_s, u_s, \preceq_s\rangle$ denote the corresponding annotated configuration.

Claim 1: For two nodes p, q in the same slice s, if $\mathit{state}(q) \prec_s \mathit{state}(p)$, $\mathit{state}(p) \in f_s$ then there is a path from p to a node p' in the target slice labeled by an f-state, s.t. for all nodes q' in the target slice that can be reached from q, $\mathit{state}(q') \prec \mathit{state}(p')$.
Proof by induction on the length of S using constraint (8).

Claim 2: For two nodes p', q' in the same slice s, if $\mathit{state}(p') \prec_s \mathit{state}(q')$, $\mathit{state}(p') \in f_s$ then there is a path from a source node p, with $\mathit{state}(p) \in f$, to p', s.t. for all nodes q in the source slice that can reach q', $\mathit{state}(p) \prec \mathit{state}(q)$.
Proof by induction on the length of S using constraint (7).

Claim 3: For all nodes p' in the target slice that are reachable from a source node p: $\mathit{state}(p') \preceq \mathit{state}(p)$.
Proof: Case (A): $\mathit{state}(p) \in f$. Assume there is a path from p to a node p' in the target slice with $\mathit{state}(p) \prec \mathit{state}(p')$. Let q' be the node in the target s.t. $\mathit{state}(q') = \mathit{state}(p)$. By Claim 2, there is a path from a node q in the source slice with $\mathit{state}(q) \in f$ to q' with $\mathit{state}(q) \prec \mathit{state}(p')$, $\mathit{state}(q) \in f$. Hence, $\mathit{state}(q) \prec \mathit{state}(p) = \mathit{state}(q')$. Let o' be the node in the target slice s.t. $\mathit{state}(o') = \mathit{state}(q)$. Again, using Claim 2, we can find a node in the source slice with an f-state that is smaller than $\mathit{state}(q)$. Since this argument can be repeated infinitely often the source configuration must contain infinitely many different states.
Case (B): $\mathit{state}(p) \notin f$. Let s' be the first slice with a $f_{s'}$-node p_1 on the path from p to p', and s the slice with the non-f_s predecessor p_0 of p_1. By constraint (6) there must be a f_s-predecessor p_0' of p_1, s.t. $\mathit{state}(p_0') \prec \mathit{state}(p_0)$. By Claim 2, there is a source node q with $\mathit{state}(q) \in f$ and $\mathit{state}(q) \prec \mathit{state}(p)$. By case

(A) all target nodes that are reachable from q are labeled by states smaller than or equivalent to $state(q)$. In particular, $state(p') \preceq state(q) \prec state(p)$.

Claim 4: For a node p' in some slice s with $state(p') \in f_s$ there is a path from a source node p to a target node p'' with $state(p) \approx state(p'')$ and $state(p) \in f, state(p'') \in f$ that visits p'.
Proof: An induction on the size of the segment of S up to s using constraint (2) and a second induction on the size of the segment beginning with s using constraint (3) shows that there is indeed a source node $p \in f$ and a target node $p'' \in f$ s.t. p' is on a path between them. By Claim 3, $state(p'') \preceq state(p)$. Now assume $state(p'') \prec state(p)$. By Claim 2, there is a source node $q \in f$ s.t. $state(q) \prec state(q)$. Let o be the node in the target slice labeled by $state(q)$. Again, using Claim 2, we can find a node in the source slice with an f-state smaller than $state(q)$. Since this argument can be repeated infinitely often the source configuration must contain infinitely many different states.

Claim 5: If there is a path from a source node p to a target node p'' with $state(p) \approx state(p'')$, then for all nodes p' on the path (where p' is a node in slice s), $state(p') \in f_s$.
Proof: Since all states in the source slice are contained in t, we know (because of constraint 4) that every path in S visits at least one $f_{s'}$-node in some slice s'. Consider the case that $state(p) \notin f$. Now let s' be the first slice with a $f_{s'}$-node p_1 that is visited on the path from p to p''. Let s be the previous slice containing p_0, the non-f_s predecessor of p_1. By constraint (6), there is a node q_0 in s, s.t. p_1 is a successor of q_0, $state(q_0) \in f_s$ and $state(q_0) \prec_s state(p_0)$. By Claim 2, there is a source node q s.t. $state(q) \prec state(p)$ and there is a path from q to q_0. Since p'' is reachable from q, by Claim 3, $state(p'') \preceq state(q)$. This is in contradiction to $state(p) \approx state(p'')$.
Now consider the case that $state(p) \in f$. Let s' be the first slice with a non-$f_{s'}$-node p_1 that is visited on the path from p to p''. Let s be the previous slice containing p_0, the f_s-predecessor of p_1. By constraint (8) there is a node q_1 in s', s.t. p_0 is a predecessor of p_1, $state(q_1) \in f_{s'}$, and $state(p_1) \prec_{s'} state(q_1)$. By Claim 1, there is a target node q'' s.t. $state(m'') \prec state(q'')$, and there is a path from q_1 to q''. Since q'' is reachable from p, by Claim 3 $state(q'') \preceq state(p)$. This again is in contradiction to $state(p) \approx state(p'')$.

Proof of the lemma: By Claims 4 and 5, the fixed nodes are exactly the nodes labeled by f_s-states. Because of $u = x$ and constraint 5 all paths in S visit an α-node. By Claim 3, all paths lead to smaller or equivalent states in the target. Paths that visit a non-fixed node lead to target nodes with strictly smaller states by Claim 5. □

With these results we can now formulate the algorithm for the emptiness check of alternating automata, shown in Figure 5. Let REACHABLE($\mathcal{A}$) denote the set of reachable configurations. ANNOTATE(X) computes for a set of configurations X a set of annotated configurations, s.t. for a configuration x all annotations $\langle x, f, f, x \cap \alpha, \preceq \rangle$ are added where $f \subseteq x \cap \beta$. We state the correctness of the algorithm as the following two theorems.

Theorem 5. *If* $\mathcal{L}(\mathcal{A}) = \emptyset$ *then* EMPTY($\mathcal{A}$).

```
EMPTY($\mathcal{A}$)
1 $A \leftarrow \emptyset$
2 $B \leftarrow$ ANNOTATE(REACHABLE($\mathcal{A}$))
3 while $(A \neq B)$ do
4     $A \leftarrow B$
5     $B \leftarrow B \cap$ FILTER(EVENTUALACCEPT(UNMARK($B$)))
6 return $(B = \emptyset)$
```

Fig. 5. Fixpoint algorithm for the emptiness check of alternating automata.

Proof. Suppose there is an annotated configuration $\langle x, f, t, u, \preceq \rangle \in B$. By Lemma 4 there exists a gratifying segment L leading from configuration x to x. Since $x \in$ REACHABLE($\mathcal{A}$) there is a run segment P leading from an initial configuration to x. Thus, by Lemma 1, $\mathcal{A}$ has a computation $P \cdot L^{\omega}$. □

Theorem 6. *If* EMPTY($\mathcal{A}$) *then* $\mathcal{L}(\mathcal{A}) = \emptyset$.

Proof. Suppose there is a computation G of $\mathcal{A}$. By Lemma 2, there is an initial segment P and an infinitely often repeated gratifying segment L. Let x be the source configuration of L. $x \in$ REACHABLE($\mathcal{A}$). By Lemma 3 there is an annotated configuration $a = \langle x, f, t = x, u = x, \preceq \rangle$ that is included in EVENTUALACCEPT(Y), if $a' = \langle x, f, t' = f, u' = x \cap \alpha, \preceq \rangle \in Y$. Since $a \in$ ANNOTATE(REACHABLE($\mathcal{A}$)), $a' \in$ UNMARK(Y) if $a \in Y$, and $a \in$ FILTER(Y) if $a \in Y$, a is included in every iteration of B. □

7 Conclusions

The data structures and algorithms presented in this paper are the basis of a symbolic verification system for language containment. In comparison to the classic construction, that starts with the determinization of the specification automaton, our algorithm is both simpler and, for certain problems, more efficient: because the two automata are combined early, no effort is wasted on the determinization of parts of the specification automaton that are not reachable in the intersection with the implementation automaton.

It should be noted, however, that our solution does not improve on the worst-case complexity of the standard algorithm. While first results with our prototype implementation are encouraging, advanced implementations and case studies are necessary to determine the characteristics of systems for which the symbolic approach is useful. The performance of NBDDs depends strongly on implementation issues like the constraints of the chosen normal form.

Efficient representations of sets of Boolean functions are of interest beyond the language containment problem. An example is the state minimization of incompletely specified finite state machines [KVBSV94]: the standard algorithm computes sets of sets of (compatible) states.

Acknowledgements: I am grateful to the members of the STeP research group at Stanford University for our discussions and their comments on drafts of this paper, as well as to the anonymous referees for their comments and suggestions.

References

[AHM+98] R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. Mocha: modularity in model checking. In A. Hu and M. Vardi, editors, *CAV 98: Computer-aided Verification*, Lecture Notes in Computer Science 1427, pages 521–525. Springer-Verlag, 1998.

[BD96] V. Bertacco and M. Damiani. Boolean function representation using parallel access diagrams. In *The Sixth Great Lakes Symposium on VLSI*. IEEE, 1996.

[BMUV97] N. Buhrke, O. Matz, S. Ulbrand, and J. Vöge. The automata theory package omega. In *WIA'97*, vol. 1436 of *LNCS*. Springer-Verlag, 1997.

[Bry86] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[CGP99] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[KV97] O. Kupferman and M. Vardi. Weak alternating automata are not that weak. In *5th Israeli Symposium on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society Press, 1997.

[KVBSV94] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. A fully implicit algorithm for exact state minimization. In *31st ACM/IEEE Design Automation Conference*, pages 684–690. ACM, 1994.

[MS87] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54(2–3):267–276, October 1987.

[MS00] Z. Manna and H.B. Sipma. Alternating the temporal picture for safety. In U. Montanari, J.D. Rolim, and E. Welzl, editors, *Proc. 27th Intl. Colloq. Aut. Lang. Prog.*, vol. 1853, pages 429–450, Geneva, Switzerland, July 2000. Springer-Verlag.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. Found. of Comp. Sci.*, pages 319–327, 1988.

[TBK95] H. Touati, R.K. Brayton, and R. Kurshan. Testing language containment for ω-automata using BDDs. *Inf. and Comp.*, 118(1):101–109, April 1995.

[THB95] S. Tasiran, R. Hojati, and R.K. Brayton. Language containment using non-deterministic omega-automata. In *Proc. of CHARME '95: Advanced Research Working Conference on Correct Hardware design and verification methods*, vol. 987 of *LNCS*. Springer-Verlag, 1995.

[Tho94] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier Science Publishers (North-Holland), 1994.

[TNB+97] K. Takagi, K. Nitta, H. Bouno, Y. Takenaga, and S. Yajima. Computational power of nondeterministic ordered binary decision diagrams and their subclasses. *IEICE Transactions on Fundamentals*, E80-A(4):663–669, April 1997.

[Var95] M.Y. Vardi. Alternating automata and program verification. In J. van Leeuwen, editor, *Computer Science Today. Recent Trends and Developments*, vol. 1000 of *LNCS*, pages 471–485. Springer-Verlag, 1995.