Decision Procedures for Inductive Boolean Functions based on Alternating Automata

Abdelwaheb Ayari, David Basin, and Felix Klaedtke

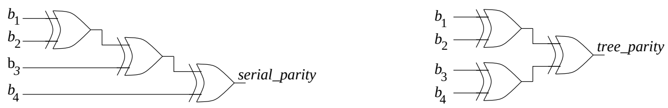
Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Germany.

Abstract. We show how alternating automata provide decision procedures for the equivalence of inductively defined Boolean functions that are useful for reasoning about parameterized families of circuits. We use alternating word automata to formalize families of linearly structured circuits and alternating tree automata to formalize families of tree structured circuits. We provide complexity bounds and show how our decision procedures can be implemented using BDDs. In comparison to previous work, our approach is simpler, yields better complexity bounds, and, in the case of tree structured families, is more general.

1 Introduction

Reasoning about parametric system descriptions is important in building scalable systems and generic designs. In hardware verification, the problem arises in verification of parametric combinational circuit families, for example, proving that circuits in one family are equivalent to circuits in another, for every parameter value. Another application of parametric reasoning is in establishing properties of sequential circuits, where time is the parameter considered. In this paper we present a new approach to these problems based on alternating automata on words and trees.

The starting point for our research is the work of Gupta and Fisher [6, 7]. They developed a formalism for describing circuit families using one of two kinds of inductively defined Boolean functions. The first, called *Linearly Inductive Boolean Functions*, or LIFs, formalizes families of linearly structured circuits. The second, called *Exponentially Inductive Boolean Functions*, or EIFs, models families of tree structured circuits. As simple examples, consider the linear (serial) and tree structured 4-bit parity circuits described by the following diagrams.



A LIF describing the general case of the linear circuit is given by the following equations. (We will formally introduce slightly different syntax in §3 and §4.)

$$\begin{aligned} \mathit{serial\_parity}^1(b_1) &= b_1 \\ \mathit{serial\_parity}^n(b_1,\ldots,b_n) &= b_n \oplus \mathit{serial\_parity}^{n-1}(b_1,\ldots,b_{n-1}) \qquad \text{for } n>1\ . \end{aligned}$$

Similarly, an EIF describing the family of tree-structured parity circuits is:

$$\begin{aligned} \mathit{tree\_parity}^1(b_1) &= b_1 \\ \mathit{tree\_parity}^{2^n}(b_1,\ldots,b_{2^n}) &= \mathit{tree\_parity}^{2^{n-1}}(b_1,\ldots,b_{2^{n-1}}) \oplus \\ &\quad\ \mathit{tree\_parity}^{2^{n-1}}(b_{2^{n-1}+1},\ldots,b_{2^n}) \qquad \text{for } n \geq 1 \ . \end{aligned}$$

Gupta and Fisher developed algorithms to translate these descriptions into novel data-structures that generalize BDDs (roughly speaking, their data-structures have additional pointers between BDDs, which formalize recursion). The resulting data-structures are canonical: different descriptions of the same family are converted into identical data-structures. This yields a decision procedure both for the equivalence of LIFs and for EIFs.

Motivated by their results, we take a different approach. We show how LIFs and EIFs can be translated, respectively, into alternating word and tree automata, whereby the decision problems for LIFs and EIFs are solvable by automata calculations. For LIFs, the translation and decision procedure are quite direct and may be implemented and analyzed using standard algorithms and results for word automata. For EIFs, the situation is more subtle since input is given by trees where only leaves are labeled by data and we are only interested in the equality of complete trees. Here, we decide equality using a procedure that determines whether a tree automaton accepts a complete leaf-labeled tree.

The use of alternating automata has a number of advantages. First, it gives us a simple view of (and leads to simpler formalisms for) LIFs and EIFs based on standard results from automata theory. For example, the expressiveness of these languages trivially falls out of our translations: LIFs describe regular languages on words and EIFs describe regular languages on trees (modulo the subtleties alluded to above). Hence, LIFs and EIFs can formalize any circuit family whose behavior is regular in the language theoretic sense. Second, it provides a handle on the complexity of the problems. For LIFs we show that the equality problem is PSPACE-complete and for EIFs it is in EXPSPACE. The result for LIFs represents a doubly exponential improvement over the previous results of Gupta and Fisher and our results for EIFs, are to our knowledge, the first published bounds for this problem. Finally, the use of alternating automata provides a basis for adapting data-structures recently developed in the MONA project [10]; in their work, as well as ours, BDDs are used to represent automata and can often exponentially compress the representation of the transition function. We show that the use of BDDs to represent alternating automata offers similar advantages and plays an important rôle in the practical use of these techniques.

We proceed as follows. In §2 we provide background material on word and tree automata. In §3 and §4 we formalize LIFs and EIFs and explain our decision procedures. In §5 we make comparisons and in §6 draw conclusions and discuss future work.

2 Background

Boolean Logic The set $B(V)$ of *Boolean formulae (over V)* is built from the constants 0 and 1, variables $v \in V$, and the connectives $\neg$, $\vee$, $\wedge$, $\leftrightarrow$ and $\oplus$. For $\beta \in B(V)$, $\beta[\alpha_1/v_1, \ldots, \alpha_n/v_n]$ denotes the formula where the $v_i \in V$ are simultaneously replaced by the formulae $\alpha_i \in B(V)$.

Boolean formulae are interpreted in the set $\mathbb{B} = \{0, 1\}$ of *truth values*. A *substitution* is a function $\sigma : V \to \mathbb{B}$ that is homomorphically extended to $B(V)$. For $\sigma : V \to \mathbb{B}$ and $\beta \in B(V)$ we write $\sigma \models \beta$ if $\sigma(\beta) = 1$. We will sometimes identify a subset M of V with the substitution $\sigma_M : V \to \mathbb{B}$, where $\sigma_M(v) = 1$ iff $v \in M$. For example, for the formula $v_1 \oplus v_2$, we have $\{v_1\} \models v_1 \oplus v_2$ but $\{v_1, v_2\} \not\models v_1 \oplus v_2$.

Words and Trees Σ^* is the set of all words over the alphabet Σ. We write λ for the empty word and Σ^+ for $\Sigma^* \setminus \{\lambda\}$. For $u, v \in \Sigma^*$, $u.v$ denotes concatenation, $|u|$ denotes u's length, and u^{R} denotes the reversal of u.

A *Σ-labeled tree* (with branching factor $r \in \mathbb{N}$) is a function t where the range of t is Σ and the domain of t, $\mathrm{dom}(t)$ for short, is a finite subset of $\{0, \ldots, r-1\}^*$ where (i) $\mathrm{dom}(t)$ is prefix closed and (ii) if $u.i \in \mathrm{dom}(t)$, then $u.j \in \mathrm{dom}(t)$ for all $j < i$. The elements of $\mathrm{dom}(t)$ are called *nodes* and $\lambda \in \mathrm{dom}(t)$ is called the *root*. The node $u.i \in \mathrm{dom}(t)$ is a *successor* of u. A node is an *inner node* if it has successors and is a *leaf* otherwise. The *height* of t is $|t| = \max(\{0\} \cup \{|u| + 1 \mid u \in \mathrm{dom}(t)\})$. The *depth* of a node $u \in \mathrm{dom}(t)$ is the length of u.

A tree is *complete* if all its leaves have the same depth. The *frontier* of t is the word $\mathrm{front}(t) \in \Sigma^*$ where the ith letter is the label of the ith leaf in t (from the left). $\Sigma^{\mathrm{T}*}$ denotes the set of all binary Σ-labeled trees and $\Sigma^{\mathrm{T}+}$ is $\Sigma^{\mathrm{T}*}$ without the empty tree.

Nondeterministic Automata A *nondeterministic word automaton (NWA)* $\mathcal{A}$ is a tuple $(\Sigma, Q, q_0, F, \delta)$, where Σ is a nonempty finite alphabet, Q is a nonempty finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of accepting states, and $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is a transition function. A *run* of $\mathcal{A}$ on a word $w = a_1 \ldots a_n \in \Sigma^*$ is a word $\pi = s_1 \ldots s_{n+1} \in Q^+$ with $s_1 = q_0$ and $s_{i+1} \in \delta(s_i, a_i)$ for $1 \leq i \leq n$. π is *accepting* if $s_{n+1} \in F$. A word w is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w; $L(\mathcal{A})$ denotes the set of accepted words.

Nondeterministic (top-down, binary) tree automata (NTA) are defined analogously: $\mathcal{A}$ is a tuple $(\Sigma, Q, q_0, F, \delta)$, where Σ, Q, q_0 and F are as before. The transition function is $\delta : Q \times \Sigma \to \mathcal{P}(Q \times Q)$. A *run* of a NTA $\mathcal{A}$ on a tree $t \in \Sigma^{\mathrm{T}*}$ is a tree $\pi \in Q^{\mathrm{T}+}$, where $\mathrm{dom}(\pi) = \{\lambda\} \cup \{w.b \mid w \in \mathrm{dom}(t) \text{ and } b \in \{0, 1\}\}$. Moreover, $\pi(\lambda) = q_0$ and for $w \in \mathrm{dom}(t)$, $(\pi(w.0), \pi(w.1)) \in \delta(\pi(w), t(w))$. The run π is *accepting* if $\pi(w) \in F$ for any leaf $w \in \mathrm{dom}(\pi)$. A tree t is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on t; $L(\mathcal{A})$ denotes the set of accepted trees.

NWAs and NTAs recognize the regular word and tree languages and are effectively closed under intersection, union, complement and projection. For a detailed account of regular word and tree languages see [11] and [4] respectively.

Alternating Automata *Alternating automata* for words were introduced in [2, 3] and for trees in [15]. We use the definition of alternating automata for words from [17] and generalize it to trees. For this we need the notion of the *positive Boolean formulae*: Let $B^+(V)$ be the set of Boolean formulae built from 0, 1, $v \in V$, and the connectives $\vee$ and $\wedge$.

An *alternating word automaton (AWA)* is of the form $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ where everything is as before, except for the transition function $\delta : Q \times \Sigma \to B^+(Q)$. The same holds for *alternating tree automata (ATA)* where the only difference is the transition function $\delta : Q \times \Sigma \to B^+(Q \times \{L, R\})$. We write q^X for $(q, X) \in Q \times \{L, R\}$.

We will only define a run for an ATA; the restriction to AWA is straightforward. For an ATA, a *run* π of $\mathcal{A}$ on $t \in \Sigma^{\mathrm{T}*}$ is a $Q \times \{0,1\}^*$-labeled tree, with $\pi(\lambda) = (q_0, \lambda)$. Moreover, for each node $w \in \mathrm{dom}(\pi)$, with $\pi(w) = (q, u)$, and for all of the $r \in \mathbb{N}$ successor nodes of w:

$$\begin{aligned}&\{p^L \mid \pi(w.k) = (p, u.0) \text{ for } 0 \leq k < r\} \cup \\ &\{p^R \mid \pi(w.k) = (p, u.1) \text{ for } 0 \leq k < r\} \models \delta(q, t(u)).\end{aligned}$$

π is *accepting* if for every leaf w in π, where $\pi(w) = (p, u.k)$, u is leaf in t and $p \in F$. The *tree language accepted by* $\mathcal{A}$ is $L(\mathcal{A}) = \{t \in \Sigma^{\mathrm{T}*} \mid \mathcal{A} \text{ accepts } t\}$. If there exists an accepting run of $\mathcal{A}' = (\Sigma, Q, q, F, \delta)$ for $q \in Q$ on t, then we say that $\mathcal{A}$ *accepts* t *from* q. We use the same terminology for AWAs.

It is straightforward to construct an alternating automaton from a nondeterministic automaton of the same size. Conversely, given an AWA one can construct an equivalent NWA with at most exponentially more states [2, 3, 17]. The states of the nondeterministic automaton are the interpretations of the Boolean formulae of the alternating automaton's transition function. This construction can be generalized to tree automata. Hence alternation does not increase the expressiveness of word and tree automata but, as we will see, it does enhance their ability to model problems.

3 Linearly Inductive Boolean Functions

3.1 Definition of LIFs

We now define linearly inductive Boolean functions. Our definition differs slightly from [6–8], however they are equivalent (see §5).

Syntax Let the two sets $V = \{v_1, \ldots, v_r\}$ and $F = \{f_1, \ldots, f_s\}$ be fixed for the remainder of this paper.

A *LIF formula* (over V and F) is a pair (α, β), with $\alpha \in B(V)$ and $\beta \in B(V \uplus F)$. The formulae α and β formalize the base and step case of a recursive definition. A *LIF system* (over V and F) is a pair (F, η) where F is a set of LIF formulae over V and F and $\eta : F \to \mathsf{F}$. That is, η assigns each $f \in F$ a LIF formula $(\alpha, \beta) \in \mathsf{F}$. We will write (α_f, β_f) for $\eta(f) = (\alpha, \beta)$ and omit V and F when they are clear from the context.

Semantics Let $\mathcal{S}$ be a LIF system. An *evaluation* of $\mathcal{S}$ on $w = b_1 \dots b_n \in (\mathbb{B}^r)^+$ is a word $d_1 \dots d_n \in (\mathbb{B}^s)^+$ such that for $b_i = (a_1^i, \dots, a_r^i)$, $d_i = (c_1^i, \dots, c_s^i)$, and $1 \leq k \leq s$:

$$c_k^1 = 1 \qquad \text{iff} \qquad \{v_l \mid a_l^1 = 1, \text{ for } 1 \leq l \leq r\} \models \alpha_{f_k} ,$$

and for all i, $1 < i \leq n$,

$$\begin{aligned} c_k^i = 1 \qquad \text{iff} \qquad & \{v_l \mid a_l^i = 1, \text{ for } 1 \leq l \leq r\} \cup \\ & \{f_l \mid c_l^{i-1} = 1, \text{ for } 1 \leq l \leq s\} \models \beta_{f_k} . \end{aligned}$$

An easy induction over the length of w shows:

Lemma 1. *For $\mathcal{S}$ a LIF system and $w \in (\mathbb{B}^r)^+$, the evaluation of $\mathcal{S}$ on w is uniquely defined.*

Hence $f_k \in F$ together with $\mathcal{S}$ determine a function $f_k^{\mathcal{S}} : (\mathbb{B}^r)^+ \to \mathbb{B}$. Namely, for $w \in (\mathbb{B}^r)^+$, $f_k^{\mathcal{S}}(w) = c_k^{|w|}$. We call $f_k^{\mathcal{S}}$ the *LIF* of $\mathcal{S}$ and f_k. When $\mathcal{S}$ is clear from the context, we omit it.

Examples We present three simple examples. First, for $V = \{x\}$ and $F = \{\mathit{serial\_parity}\}$, the following LIF system $\mathcal{S}_1$ formalizes the family of linear parity circuits given in the introduction.

$$\alpha_{serial\_parity} = x \qquad\qquad \beta_{serial\_parity} = x \oplus serial\_parity$$

In particular, $\mathit{serial\_parity}^{\mathcal{S}_1}$ applied to $b_1 \dots b_n \in \mathbb{B}^+$ equals $\mathit{parity}^n(b_1, \dots, b_n)$.

The second LIF system $\mathcal{S}_2$ over $V = \{a, b, cin\}$ and $F = \{sum, carry\}$ formalizes a family of ripple-carry adders.

$$\begin{aligned} \alpha_{sum} &= (a \oplus b) \oplus cin & \beta_{sum} &= (a \oplus b) \oplus carry \\ \alpha_{carry} &= (a \wedge b) \vee ((a \vee b) \wedge cin) & \beta_{carry} &= (a \wedge b) \vee ((a \vee b) \wedge carry) \end{aligned}$$

Here $\mathit{sum}^{\mathcal{S}_2}$ [respectively $\mathit{carry}^{\mathcal{S}_2}$] represents the adder's nth output bit [respectively carry bit].

The third example shows how to describe a sequential circuit by a LIF system. The LIF system $\mathcal{S}_3$ over $V = \{e\}$ and $F = \{Y_1, Y_2, Y_3\}$ describes a 3-bit counter with an enable bit.

$$\begin{aligned} \alpha_{Y_1} &= 0 & \beta_{Y_1} &= (e \wedge \neg Y_1) \vee (\neg e \wedge Y_1) \\ \alpha_{Y_2} &= 0 & \beta_{Y_2} &= (e \wedge (Y_1 \oplus Y_2)) \vee (\neg e \wedge Y_2) \\ \alpha_{Y_3} &= 0 & \beta_{Y_3} &= (e \wedge ((Y_1 \wedge Y_2) \oplus Y_3)) \vee (\neg e \wedge Y_3). \end{aligned}$$

$Y_i^{\mathcal{S}_3}(w)$, with $w \in \mathbb{B}^+$, is the value of the ith output bit at time $|w|$ of the 3-bit counter, where w encodes the enable input signals.

3.2 Equivalence of LIF Systems and AWAs

A function $g : (\mathbb{B}^r)^+ \to \mathbb{B}$ is *LIF-representable* if there exists a LIF system $\mathcal{S}$ and a $f \in F$, where $g(w) = f^{\mathcal{S}}(w)$ for all $w \in (\mathbb{B}^r)^+$. A language $L \subseteq (\mathbb{B}^r)^+$ is *LIF-representable* if its characteristic function, $g(w) = 1$ iff $w \in L$, is LIF-representable. Gupta and Fisher have shown in [6, 9] that any LIF-representable language is regular. They prove that their data-structure for representing a LIF system corresponds to a minimal deterministic automaton that accepts the language $\{w^{\mathrm{R}} \mid f^{\mathcal{S}}(w) = 1, \text{ for } w \in (\mathbb{B}^r)^+\}$.

We present here a simpler proof of regularity by showing that LIF systems directly correspond to AWAs. We also prove a weakened form of the converse: almost all regular languages are LIF-representable. The weakening though is trivial and concerns the empty word, and if we consider languages without the empty word we have an equivalence.[1] Hence, for the remainder of this section, *we consider only automata (languages) that do not accept (include) the empty word λ.*

For technical reasons we will work with LIF systems in a kind of negation normal form. A Boolean formula $\beta \in B(X)$ is *positive in* $Y \subseteq X$ if negations occur only directly in front of the Boolean variables $v \in X \setminus Y$ and, furthermore, the only connectives allowed are $\neg$, $\wedge$ and $\vee$. A LIF system $\mathcal{S}$ is in *normal form* if β_f is positive in F, for each $f \in F$.

Lemma 2. *Let $\mathcal{S}$ be a LIF system over V and F. Then there is a LIF system $\mathcal{S}'$ over V and $F' = F \uplus \{\overline{f} \mid f \in F\}$ in normal form where, for all $f \in F$ and $w \in (\mathbb{B}^r)^+$, $f^{\mathcal{S}'}(w) = f^{\mathcal{S}}(w)$ and $\overline{f}^{\mathcal{S}'}(w) = 1$ iff $f^{\mathcal{S}}(w) = 0$,*

Proof. Without loss of generality, we assume that for $\beta \in B(X)$ only the connectives $\neg$, $\vee$, and $\wedge$ occur. The other connectives can be eliminated as standard, which may lead to exponentially larger formulae. By $\mathrm{nnf}(\beta)$ we denote the negation normal form of $\beta \in B(X)$.

By using the same idea as [3], it is easy to construct a LIF system $\mathcal{S}'$ by introducing for each $f \in F$ a new variable $\overline{f}$ that "simulates" $\neg f$. Let $\mathcal{S} = (\mathsf{F}, \eta)$. For $f \in F$, with $\eta(f) = (\alpha, \beta)$, the mapping η' of the LIF system $\mathcal{S}'$ is defined by $\eta'(f) = (\alpha, \gamma)$ and $\eta'(\overline{f}) = (\neg\alpha, \overline{\gamma})$ where γ and $\overline{\gamma}$ are obtained from $\mathrm{nnf}(\beta)$, respectively $\mathrm{nnf}(\neg\beta)$, by replacing the sub-formulae $\neg f_i$ by $\overline{f}_i$. $\square$

We now prove that LIF-representable languages and (λ-free) regular languages coincide.

[1] We can easily redefine LIFs to define functions over $(\mathbb{B}^r)^*$. However, following Gupta and Fisher we avoid this as the degenerate base case (0 length input) is ill-suited for modeling parametric circuits. Ignoring the empty word is immaterial for our complexity and algorithmic analysis.

Theorem 1. *LIF systems are equivalent to AWAs. In particular:*

i) Given an AWA $\mathcal{A} = (\mathbb{B}^r, Q, q_0, F, \delta)$, there is a LIF system $\mathcal{S}$ over $V = \{v_1, \ldots, v_r\}$ and Q in normal form such that for all $w \in (\mathbb{B}^r)^+$ and $q \in Q$,

$$q^{\mathcal{S}}(w) = 1 \qquad \textit{iff} \qquad \mathcal{A} \textit{ accepts } w^{\mathrm{R}} \textit{ from } q\,.$$

ii) Given a LIF system $\mathcal{S}$ in normal form over V and F, there exists an AWA $\mathcal{A}$ with states $F \uplus \{q_{base}, q_{step}\}$ such that for all $w \in (\mathbb{B}^r)^+$ and $f \in F$,

$$\mathcal{A} \textit{ accepts } w \textit{ from } f \qquad \textit{iff} \qquad f^{\mathcal{S}}(w^{\mathrm{R}}) = 1\,.$$

Proof. (i) We encode each $b \in \mathbb{B}^r$ by a formula $\gamma_b \in B(V)$. For example, $(0,1,1,0) \in \mathbb{B}^4$ is encoded as the Boolean formula $\gamma_{(0,1,1,0)} = \neg v_1 \wedge v_2 \wedge v_3 \wedge \neg v_4$. The LIF formula for q in $\mathcal{S}$ is

$$\alpha_q = \bigvee_{b \in \mathbb{B}^r} \big(\gamma_b \wedge B(q,b)\big) \qquad\qquad \beta_q = \bigvee_{b \in \mathbb{B}^r} \big(\gamma_b \wedge \delta(q,b)\big)$$

with $B(q,b) = 1$ iff $F \models \delta(q,b)$. Here, the Boolean formula β_q simulates the transition from the state q on a non-final letter of the input word. The final state set F is simulated by the Boolean formula α_q, i.e., $F \models \delta(q,b)$ iff $\{v_i \mid b_i = 1\} \models \alpha_q$.

We prove (i) by induction over the length of $w \in (\mathbb{B}^r)^+$. If $|w| = 1$, then the equivalence follows from the definition of α_q, for any $q \in Q$. Assume (i) is true for the word w, i.e., for each $q_k \in Q$, $\mathcal{A}$ accepts w^{R} from q_k iff $q_k^{\mathcal{S}}(w) = 1$. Let $u.d$ be an evaluation of $\mathcal{S}$ on w with $d = (c_1, \ldots, c_{|Q|})$. It holds that $q_k^{\mathcal{S}}(w) = 1$ iff $c_k = 1$. We prove (i) for $w.b$ with $b = (a_1, \ldots, a_r)$. As defined, for each $q \in Q$ we have $q^{\mathcal{S}}(w.b) = 1$ iff

$$\{v_l \mid a_l = 1, \text{ for } 1 \le l \le r\} \cup \{q_l \mid c_l = 1, \text{ for } 1 \le l \le |Q|\} \models \bigvee_{b' \in \mathbb{B}^r} \big(\gamma_{b'} \wedge \delta(q,b')\big).$$

By the induction hypothesis, we obtain $\{q_l \mid \mathcal{A}$ accepts w^{R} from q_l, for $1 \le l \le |Q|\} \models \delta(q_k, b)$. From this we can easily construct an accepting run of $\mathcal{A}$ from q on $(w.b)^{\mathrm{R}}$. The other direction holds by definition of an accepting run.

(ii) For an arbitrary $g \in F$, let $\mathcal{A} = (\mathbb{B}^r, F \uplus \{q_{base}, q_{step}\}, g, \{q_{base}\}, \delta)$ with $\delta(q_{base}, b) = 0$, $\delta(q_{step}, b) = q_{base} \vee q_{step}$, and for $f \in F$

$$\delta(f, (b_1, \ldots, b_r)) = \big(q_{step} \wedge \beta_f[b_1/v_1, \ldots, b_r/v_r]\big) \vee \begin{cases} q_{base} & \text{if } \{v_i \mid b_i = 1\} \models \alpha_f \\ 0 & \text{otherwise} \end{cases}$$

Intuitively when $\mathcal{A}$ is in state $f \in F$ and reads $(b_1, \ldots, b_r) \in \mathbb{B}^r$ it guesses if the base case is reached. When this is the case, the next state is q_{base} iff $\{v_i \mid b_i = 1\} \models \alpha_f$. Otherwise, if the base case is not reached, the AWA proceeds according to the step case given by the Boolean formula β_f of the LIF system. The equivalence is proved in a similar way to (i). $\square$

Note that if a LIF formula only uses the connectives $\neg$, $\wedge$ and $\vee$, then, following the proof of Lemma 2, a normal form can be obtained in polynomial time. Moreover, if V is fixed the size, the AWA $\mathcal{A}$ of Theorem 1(ii) can be constructed in polynomial time, since the size of the alphabet $\mathbb{B}^r$ is a constant. However, if we allow V to vary, then the size of the AWA constructed can be exponentially larger than the size of the LIF system, i.e. $|V|+|F|+\sum_{f\in F}(|\alpha_f|+|\beta_f|)$, since the input alphabet of $\mathcal{A}$ is of size $2^{|V|}$.

3.3 Deciding LIF Equality

Given LIF systems $\mathcal{S}$ over V and F, and $\mathcal{T}$ over V and G, and function symbols $f_k \in F$ and $g_l \in G$, the *equality problem for LIFs* is to decide whether $f_k^{\mathcal{S}}(w) = g_l^{\mathcal{T}}(w)$, for all $w \in (\mathbb{B}^r)^+$. We first show that this problem is PSPACE-complete and afterwards show how, using BDDs, the construction in Theorem 1 provides the basis for an efficient implementation.

Theorem 2. *The equality problem for LIFs is PSPACE-complete.*

Proof. We reduce the emptiness problem for AWAs, which is PSPACE-hard [12, 17], to the equality problem for LIFs. Given an AWA $\mathcal{A}$ with initial state q_0, by Theorem 1(i) we can construct an equivalent LIF system $\mathcal{S}$ in polynomial time. Let the LIF system $\mathcal{T}$ be given by the formulae $\alpha_g = 0$ and $\beta_g = 0$. Then $q_0^{\mathcal{S}} = g^{\mathcal{T}}$ iff $L(\mathcal{A}) = \emptyset$.

Theorem 1(ii) cannot be used to show that the problem is in PSPACE because, as explained in the previous section, both the normal form and the size of the two constructed AWAs can be exponentially larger than the size of the LIF instances. Hence, we instead give a direct proof. The following Turing machine $\mathcal{M}$ accepts a problem instance in PSPACE iff a word $w = b_1 \dots b_n \in (\mathbb{B}^r)^+$ exists with $f_k^{\mathcal{S}}(w) \neq g_l^{\mathcal{T}}(w)$. Let $d_1 \dots d_n \in (\mathbb{B}^{|F|})^+$ be the evaluation of $\mathcal{S}$ on w and $d'_1 \dots d'_n \in (\mathbb{B}^{|G|})^+$ be the evaluation of $\mathcal{T}$ on w. $\mathcal{M}$ guesses in the ith step $b_i \in \mathbb{B}^r$ and calculates $d_i = (c_1, \dots, c_{|F|})$ and $d'_i = (c'_1, \dots, c'_{|G|})$ of the evaluations. If $c_k \neq c'_l$ then $\mathcal{M}$ accepts the instance and otherwise $\mathcal{M}$ continues with the $(i+1)$th step. Note that for the ith step only d_{i-1} and d'_{i-1} and b_i are required to calculate d_i and d'_i. Hence $\mathcal{M}$ runs in polynomial space, since in the ith step the space $|V|$ is required to store b_i and the space $2(|F|+|G|)$ to store d_{i-1}, d_i, d'_{i-1}, and d'_i. $\mathcal{M}$ needs linear time in the size of the LIF formulae of $\mathcal{S}$ and $\mathcal{T}$ to calculate d_i and d'_i from b_i, d_{i-1} and d'_{i-1}. Since PSPACE is closed under nondeterminism and complementation, the equality problem for LIFs is in PSPACE. $\square$

Although the machinery of alternating automata may appear a bit heavy, it leads to simple translations as there is a direct correspondence between function symbols in a LIF system and states in the corresponding AWA. This would not be possible using nondeterministic automata. Because the emptiness problem for NWAs is LOGSPACE-complete and the equality problem for LIFs is PSPACE-complete, a translation of a LIF system to a nondeterministic automata must, in general, lead to an exponential blow-up in the state space.

INPUT: AWA $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$
OUTPUT: returns *true* iff $L(\mathcal{A}) = \emptyset$

Current := $\{\{q_0\}\}$;
Processed := $\emptyset$;
while *Current* $\neq \emptyset$ **do begin**
  if *Current* $\cap \mathcal{P}(F) \neq \emptyset$ **then return** *false*;
  else begin
    Processed := *Processed* $\cup$ *Current*;
    Current := $\{T' \subseteq Q \,|\, T' \models \bigwedge_{q\in T} \delta(q,a)$ for $T \in$ *Current*, $a \in \Sigma\} \setminus$ *Processed*;
  end;
end;
return *true*;

Fig. 1. Decision procedure for the emptiness problem for AWAs.

Implementation In the proof of Theorem 2 we did not use the mapping from LIFs to AWAs given by Theorem 1(ii) due to the possible exponential blow-up when normalizing the LIF system, and the certain exponential blow-up in representing the AWA's alphabet. We describe here how these blow-ups can sometimes be avoided by using BDDs.

The reduction of LIF equality to the emptiness problem for AWAs is straightforward. From the LIF systems $\mathcal{S}$ over V and F, and $\mathcal{T}$ over V and G we construct the LIF system $\widetilde{\mathcal{S}}$ over V and $\{\widetilde{f}\} \uplus F \uplus G$ with the additional LIF formula $\alpha_{\widetilde{f}} = \neg(\alpha_f \leftrightarrow \alpha_g)$ and $\beta_{\widetilde{f}} = \neg(\beta_f \leftrightarrow \beta_g)$. We then normalize $\widetilde{\mathcal{S}}$ and use Theorem 1(ii) to construct the AWA $\mathcal{A}$ with the initial state $\widetilde{f}$. By construction, $L(\mathcal{A}) \neq \emptyset$ iff $\widetilde{f}^{\widetilde{\mathcal{S}}}(w) = 1$ for some $w \in (\mathbb{B}^r)^+$ iff $f^{\mathcal{S}} \neq g^{\mathcal{T}}$.

To decide if an AWA $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ accepts the empty language, we construct "on-the-fly" the equivalent NWA $\mathcal{B} = (\Sigma, \mathcal{P}(Q), \{q_0\}, \mathcal{P}(F), \delta')$ with

$$\delta'(T,a) = \{T' \subseteq Q \mid T' \models \bigwedge_{q\in T} \delta(q,a)\},$$

and search for a path from the initial state $\{q_0\}$ of $\mathcal{B}$ to a final state. We do this with a parallel breadth-first search in the state space of $\mathcal{B}$ as described in Figure 1.

To analyze the complexity, observe that the **while**-loop is traversed maximally $2^{|Q|}$-times and the calculation in each iteration requires $O(2^{|Q|}|\Sigma|)$-time. Hence the worst-case running time is $O(2^{2|Q|}|\Sigma|)$. We need two vectors of the length $2^{|Q|}$ to represent the sets *Current* and *Processed*. Hence the required space is the maximum of $O(2^{|Q|})$ and the size of the representation of the AWA $\mathcal{A}$.

It is possible to use BDDs in two places to sometimes achieve an exponential savings in space. First, the sets *Current*, *Processed* $\subseteq \mathcal{P}(Q)$ can be encoded as BDDs where a BDD represents the characteristic function of the set. Second, since the size of $\mathcal{A}$'s alphabet ($\mathbb{B}^r$) is exponential in $|V|$, we use the same idea that Gupta and Fisher employed for their representation of LIFs: we need not

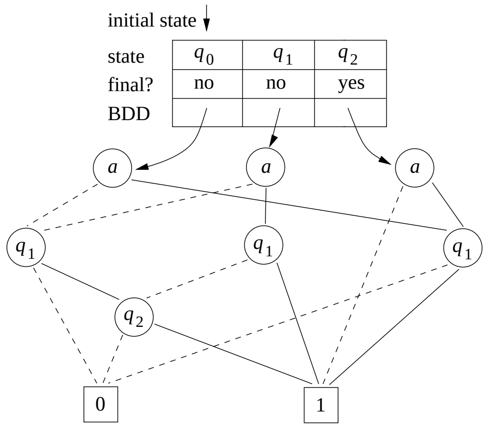


Fig. 2. Representation of an AWA.

explicitly represent the exponentially large alphabet and we can use BDDs to represent the transition function. For example, Figure 2 depicts the representation of the AWA $\mathcal{A} = (\mathbb{B}, \{q_0, q_1, q_2\}, q_0, \{q_2\}, \delta)$ with the transition function

$$\begin{aligned} \delta(q_0, 0) &= q_1 \wedge q_2 & \qquad \delta(q_0, 1) &= q_1 \\ \delta(q_1, 0) &= q_1 \wedge q_2 & \qquad \delta(q_1, 1) &= q_1 \vee q_2 \\ \delta(q_2, 0) &= 1 & \qquad \delta(q_2, 1) &= q_1 . \end{aligned}$$

The solid [respectively dashed] lines correspond to the variable assignment 1 [respectively 0]. For example, the state q_0 has a pointer to a BDD, where the first node (labeled a) encodes the alphabet; the solid line from this node points to a BDD representing $\delta(q_0, 1) = q_1$ and the dashed line points to a BDD representing $\delta(q_0, 0) = q_1 \wedge q_2$.

We have implemented the emptiness test for AWAs using the CUDD package [16] and have begun preliminary testing and comparison. For the examples given in §3.1, building AWAs for the descriptions given and testing them for emptiness or equivalence with alternative descriptions is very fast: it takes a fraction of a second and most of the time is spent with I/O. We can carry out more ambitious tests by scaling up the sequential 3-bit counter example, namely performing tests on an n-bit counter for different values of n. This example is also interesting as it demonstrates the worst-case performance of our decision procedure since exponential many states of the NWA must be constructed to decide if the AWA describes the empty language.

Table 1 gives empirical results of the required space and time for the emptiness test for the resulting AWAs on a SUN Sparc Ultra with 250MHz. In the rightmost column are the running times on a SUN Sparc Ultra with 300MHz for building the canonical representation in Gupta and Fisher's approach. For large values of n, our approach yields significantly better results, although in both cases the algorithms require exponential time and space. Note that some

n	# BDD nodes of transition function	peak of # BDD nodes of *Current*/*Processed*	AWA CPU time	LIF CPU time
2	21	10 / 11	$0.1s$	$0.0s$
4	59	34 / 42	$0.1s$	$0.0s$
8	189	318 / 717	$1.0s$	$4.0s$
10	269	971 / 3063	$8.2s$	$81.0s$
12	371	3438 / 13037	$71.7s$	$15241.5s$

Table 1. Empirical results of the emptiness test for a n-bit counter

care must be taken in comparing these results: we have not included the time taken in constructing the AWA from the LIF system (it is linear) as this was done by hand. Further, Gupta and Fisher build a canonical representation of the LIF and they have used the older BDD package from David Long.

4 Exponentially Inductive Boolean Functions

The structure of this section parallels that of §3. After defining EIFs, we show how their equality problem can be decided using tree automata. The decision procedure however is not as direct as it is for LIFs. One problem is that inputs to EIFs are words not trees. We solve this by labeling the interior nodes of trees with a dummy symbol. However, the main problem is that the words must be of length 2^n. This restriction cannot be checked by tree automata and we solve this by deciding separately if a tree automaton accepts a complete tree.

4.1 Definitions of EIFs

Syntax An *EIF formula* (over V and F) is a pair (α,β), with $\alpha \in B(V)$ and $\beta \in B(F \times \{L,R\})$. We write f^L [respectively f^R] for the variable (f,L) [respectively (f,R)] in $F \times \{L,R\}$. An *EIF system* (over V and F) is a pair (F,η), where F and η are defined as for a LIF system. Similarly to LIF systems, we write (α_f,β_f) for $\eta(f)=(\alpha,\beta)$.

Semantics Let $\mathcal{S}$ be an EIF system over V and F. An *evaluation* of $\mathcal{S}$ on a word $w = b_1 \dots b_{2^n} \in (\mathbb{B}^r)^+$ is a complete binary $\mathbb{B}^s$-labeled tree τ with $\text{front}(\tau) = d_1 \dots d_{2^n} \in (\mathbb{B}^s)^+$ and for $1 \le k \le s$:

i) For $b_i = (a_1^i,\dots,a_r^i)$ and $d_i = (c_1^i,\dots,c_s^i)$, with $1 \le i \le 2^n$,
$$c_k^i = 1 \quad \text{iff} \quad \{v_l \mid a_l^i = 1, \text{ for } 1 \le l \le r\} \models \alpha_{f_k}\,.$$

ii) For each inner node $u \in \text{dom}(\tau)$ with $\tau(u.0) = (c_1',\dots,c_s')$, $\tau(u.1) = (c_1'',\dots,c_s'')$, and $\tau(u) = (c_1,\dots,c_s)$:
$$\begin{aligned} c_k = 1 \quad \text{iff} \quad & \{f_i^L \mid c_l' = 1, \text{ for } 1 \le l \le s\} \,\cup \\ & \{f_i^R \mid c_l'' = 1, \text{ for } 1 \le l \le s\} \models \beta_{f_k}\,. \end{aligned}$$

Let $\Sigma^{2+} = \{w \in \Sigma^* \mid |w| = 2^n, \text{for some } n \in \mathbb{N}\}$. As with LIFs, the evaluation τ is uniquely defined; hence $f_k \in F$ and $\mathcal{S}$ together define a function $f_k^{\mathcal{S}} : (\mathbb{B}^r)^{2+} \to \mathbb{B}$. Namely $f_k^{\mathcal{S}}(w) = c_k$, where τ is the unique evaluation of $\mathcal{S}$ on w and $\tau(\lambda) = (c_1, \ldots, c_s)$. *EIF-representable* is defined analogously to LIF-representable.

For example, the tree implementation of the parameterized parity circuit from the introduction is described by the EIF system $\mathcal{S}$ over $V = \{x\}$ and $F = \{\mathit{tree\_parity}\}$ with the EIF formula:

$$\alpha_{\mathit{tree\_parity}} = x \qquad\qquad \beta_{\mathit{tree\_parity}} = \mathit{tree\_parity}^L \oplus \mathit{tree\_parity}^R.$$

Here the value of the EIF $\mathit{tree\_parity}^{\mathcal{S}}$ applied to a word $w = b_1 \ldots b_{2^n} \in \mathbb{B}^+$ is the value of the function parity^{2^n} applied to $(b_1, \ldots, b_{2^n})$.

4.2 Equivalence of EIF Systems and ATAs

Using ATAs we can characterize the EIF-representable functions. To interpret a word in the domain of an EIF as a tree, we identify a word $b_1 \ldots b_{2^n} \in \Sigma^*$ with the complete tree $t \in (\Sigma \uplus \{\#\})^{\mathrm{T}*}$, where $\mathrm{front}(t) = b_1 \ldots b_{2^n}$ and all inner nodes are labeled with the dummy symbol $\#$. In the following, $\Sigma_\#$ stands for $\Sigma \uplus \{\#\}$. We call a tree $t \in \Sigma_\#^{\mathrm{T}*}$ a *Σ-leaf-labeled* tree when (i) if w is a leaf, then $t(w) \in \Sigma$ and (ii) if w is an inner node, then $t(w) = \#$.

Normal forms for EIF systems can be defined and obtained as for LIF systems and the proof of Theorem 1 can, with minor modifications, be generalized to EIFs.

Theorem 3. *EIF systems are equivalent to ATAs if the input trees are restricted to complete, leaf-labeled trees. In particular:*

i) Let $\mathcal{A} = (\mathbb{B}^r_\#, Q, q_0, F, \delta)$ be an ATA. There is a normal form EIF system $\mathcal{S}$ over $V = \{v_1, \ldots, v_r\}$ and Q, such that for all $q \in Q$ and any complete $\mathbb{B}^r$-leaf-labeled tree $t \in (\mathbb{B}^r_\#)^{T+}$,

$$q^{\mathcal{S}}(\mathrm{front}(t)) = 1 \qquad \textit{iff} \qquad \mathcal{A} \textit{ accepts } t \textit{ from } q\,.$$

ii) Let $\mathcal{S}$ be a normal form EIF system over V and F. There is an ATA $\mathcal{A}$ with the state set $F \uplus \{q_{base}, q_{step}\}$, such that $\mathcal{A}$ accepts from $f \in F$ only $\mathbb{B}^r$-leaf-labeled trees, and

$$\mathcal{A} \textit{ accepts from } f \textit{ the complete tree } t \qquad \textit{iff} \qquad f^{\mathcal{S}}(\mathrm{front}(t)) = 1\,.$$

4.3 Deciding EIF Equality

The *equality problem for EIFs* and the *size* of an instance are defined similarly to LIFs. We cannot generalize the decision procedure from §3.3 to EIFs since we are only interested in trees of a restricted form: complete leaf-labeled binary trees. Unfortunately completeness is not a regular property, i.e. one recognizable by tree automata, and hence we cannot reduce the problem to an emptiness problem. Instead, we reduce the problem to the *complete-tree-containment problem (CTCP) for NTAs*, which is to decide whether a given NTA accepts a complete tree.

Theorem 4. *The equality problem for EIFs is in EXPSPACE.*

Proof. Let $\mathcal{S}$ over V and F, and $\mathcal{T}$ over V and G be EIF systems, and let $f \in F$ and $g \in G$ be given. Let $\widetilde{\mathcal{S}}$ be the EIF system over V and $\{\widetilde{f}\} \uplus F \uplus G$ with the additional EIF formula $\alpha_{\widetilde{f}} = \neg(\alpha_f \leftrightarrow \alpha_g)$ and $\beta_{\widetilde{f}} = \neg(\beta_f \leftrightarrow \beta_g)$. We normalize $\widetilde{S}$, and by Theorem 3(ii) construct an ATA $\mathcal{A}$ with the initial state $\widetilde{f}$, such that $f^{\mathcal{S}} \neq g^{\mathcal{T}}$ iff $\mathcal{A}$ accepts a complete tree. $\mathcal{A}$ has $2|\{\widetilde{f}\} \uplus F \uplus G| + 2$ states and the size of the alphabet $\mathbb{B}_{\#}^{|V|}$ is $2^{|V|} + 1$. From $\mathcal{A}$ we can construct an equivalent NTA $\mathcal{B}$ that has at most $O(2^{2|F|+|G|})$ many states. Hence we have reduced the equality problem for EIFs to CTCP for NTAs. The required space for the reduction is $O(2^{|V|} 2^{2|F|+|G|})$.

We now show that CTCP for NTAs is in PSPACE. For the NTA $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ we construct the AWA $\mathcal{A}' = (\{1\}, Q, q_0, F, \delta')$ with $\delta'(q, 1) = \bigvee_{a \in \Sigma} \bigvee_{(p,p') \in \delta(q,a)} (p \wedge p')$. It is easy to prove that $\mathcal{A}$ accepts a complete tree of height h iff $\mathcal{A}'$ accepts a word of length h. From this follows that CTCP for NTAs is in PSPACE because the emptiness problem for AWAs is in PSPACE [12, 17]. □

In [13], it is proved that the equality problem for EIFs and CTCP for ATAs (to decide if an ATA accepts a complete tree) are both EXPSPACE-hard. We omit the proof, which is quite technical, due to space limitations.

5 Comparisons and Related Work

Our work was motivated by that of Gupta and Fisher [6–8] and we begin by comparing our LIFs and EIFs with theirs, which we will call LIF_0 and EIF_0.

For each $n \geq 1$, a LIF_0 f is given by a Boolean function, called the n-instance of f and denoted by f^n, where $f^1 : \mathbb{B}^r \to \mathbb{B}$ and $f^n : \mathbb{B}^{r+s} \to \mathbb{B}$ for $n > 1$ (r is the number of n-instance inputs and s is the number of $(n-1)$-instance function inputs). Further it must hold that for all $m, n > 1$ the m-instance and the n-instance of f are equal, i.e. $f^m = f^n$. By means of the parity function we explain how the value of a LIF_0 is calculated. The n-instances of $\mathit{serial\_parity}^n$ (using their notation) are:

$$\begin{aligned} \mathit{serial\_parity}^1 &= b^1, \\ \mathit{serial\_parity}^n &= b^n \oplus \mathit{serial\_parity}^{n-1} \qquad \text{for } n > 1. \end{aligned}$$

The *value* of the LIF_0 *serial_parity* on the word $b_1 \ldots b_n \in (\mathbb{B}^r)^+$, written as $\mathit{serial\_parity}(b_1, \ldots, b_n)$, is the value of the 1-instance $\mathit{serial\_parity}^1$ applied to b_1 for $n = 1$. For $n > 1$, it is the value of the n-instance $\mathit{serial\_parity}^n$ applied to b_n and $\mathit{serial\_parity}(b_1, \ldots, b_{n-1})$.

The definitions of a "LIF formula" and a "LIF system" correspond to the definition of a "LIF_0". Moreover, the way the "value" of a LIF_0 is calculated corresponds to our definition of "evaluation". Hence both formalisms are equivalent. However, the algorithms, data-structures, and complexity of our approaches are completely different!

Gupta and Fisher formalize LIF_0s using a data-structure based on BDDs where terminal nodes are not just the constants 0 and 1, but also pointers to other BDDs. They then prove that each LIF system has a canonical representation that can be obtained in $O(2^{2^{|F|}}2^{|V|} + (2^{2^{|F|}})^2)$ time and space in the worst-case. In contrast, we have given a decision procedure (Theorem 2) that requires polynomial space, which is a doubly exponential improvement in space and an exponential improvement in time. Despite its worse space complexity, our algorithm based on BDD-represented AWAs may give better results in practice than our PSPACE decision procedure. This depends on whether the BDDs used require polynomial or exponential space. If the space required is polynomial, then the resulting AWA and its emptiness test require only polynomial space. In the exponential case, as there are only $2|F| + 2$ states, the emptiness test requires $O(2^{|V|+2|F|})$ time and space. This case also represents an exponential improvement over Gupta and Fisher's results, both in time and space.

An EIF_0 f has, like a LIF_0, for each $n \geq 0$, a n-instance function f^{2^n}, where $f^1 : \mathbb{B}^r \rightarrow \mathbb{B}$ and, for $n > 0$, f^{2^n} is a Boolean combination of three EIF_0s, e, g and h, i.e. $f^{2^n} : \mathbb{B}^3 \rightarrow \mathbb{B}$. Further it must hold that $f^{2^m} = f^{2^n}$ for all $m, n > 0$. The *value* of the EIF_0 f on the word $b_1 \dots b_{2^n} \in (\mathbb{B}^r)^+$, written as $f(b_1, \dots, b_{2^n})$, is the value of the 0-instance f^1 applied to b_1 if $n = 0$. For $n > 0$, it is the value of the n-instance f^{2^n} applied to the value of the EIF_0 e of the left half of the word, i.e. $e(b_1, \dots, b_{2^{n-1}})$, and to the values of the EIF_0s g and h of the right half of the word, i.e. $g(b_{2^{n-1}+1}, \dots, b_{2^n})$ and $h(b_{2^{n-1}+1}, \dots, b_{2^n})$.

EIF_0s are strictly less expressive than EIFs. Indeed, since not every Boolean function for the n-instance function of an EIF_0 (for $n > 0$) is allowed, even simple functions cannot be described by an EIF_0, e.g., $F : \mathbb{B}^{2+} \rightarrow \mathbb{B}$ with $F(w) = 1$ iff $w = 0000$ or $w = 1100$ or $w = 1011$. The reason is similar to why deterministic top-down tree automata are weaker than nondeterministic top-down tree automata; the restrictions of the n-instance function of an EIF_0 stems from the data-structure proposed for EIF_0s in [6, 7] in order to have a canonical representation. On the other hand, it is easy to see that F is EIF-representable.

Our results on the complexity of the equality problem for EIFs are, to our knowledge, the first such results given in the literature. Neither we nor Gupta and Fisher have implemented a decision procedure for the equality problem for EIFs or EIF_0s.

We have seen that LIFs and EIFs can be reduced to word and tree automata. Gupta and Fisher also give in [6, 7] an extension of their data-structure for LIF_0s and EIF_0s that handles more than one induction parameter with the restriction that the induction parameters must be mutually independent. We conjecture that it may be possible to develop similar models in our setting based on 2-

dimensional word automata (as described in [5]) and their extension to trees. However, this remains as future work.

There are also similarities between our work and the description of circuits by equations of Brzozowski and Leiss in [2]. A *system of equations* S has the form $X_i = \bigcup_{a\in\Sigma}\{a\}.F_{i,a} \cup \delta_i$ (for $1 \leq i \leq n$) where the $F_{i,a}$ are Boolean functions in the variables X_i, and each δ_i is either $\{\lambda\}$ or $\emptyset$. It is shown in [2] that a solution to S is unique and regular, i.e., if each X_i is interpreted with $L_i \subseteq \Sigma^*$ and the L_i satisfy the equations in S, then the L_i are unique and regular. LIF systems offer advantages in describing parameterized circuits. For example, with LIFs one directly describes the "input ports" using the variables V. In contrast, a system of equations must use the alphabet $\mathbb{B}^r$ and cannot "mix" input pins and the signals of the internal wiring (and the same holds for outputs). Furthermore, descriptions using LIFs cleanly separate the base and step cases of the circuit family, which is not the case with [2].

Finally, note that the use of BDDs to represent word and tree automata, without alternation, is explored in [10, 14]. There, BDD-represented automata are used to provide a decision procedures for monadic second-order logics on words and trees. This decision procedure is implemented in the MONA system, and MONA can be used to reason about LIF systems [1]: a LIF system is described by a monadic second-order formula, which MONA translates into a deterministic word automaton. Although this has the advantage of using an existing decision procedure, the complexity can be considerably worse both in theory and in practice. For example, for a 12-bit counter MONA (version 1.3) needs more than an hour to build the automaton and the number of BDD nodes is an order of magnitude larger than what is needed for our emptiness test for AWAs.

6 Conclusions

We have shown that LIFs and EIFs can be understood and analyzed using standard formalisms and results from automata theory. Not only is this conceptually attractive, but we also obtain better results for the decision problem for LIFs and the first complexity results for EIFs. The n-bit counter example in §3.3 indicates that our approach, at least in some cases, is faster in practice. However, an in depth experimental comparison of the procedures remains as future work.

Acknowledgments: The authors thank Aarti Gupta for helpful discussions and kindly providing us with recent timings using her package. We also thank the referees for their suggested improvements.

References

1. D. Basin and N. Klarlund. Automata based symbolic reasoning in hardware verification. *The Journal of Formal Methods in Systems Design*, 13(3):255–288, 1998.
2. J. Brzozowski and E. Leiss. On equations for regular languages, finite automata, and sequential networks. *TCS*, 10(1):19–35, 1980.

3. A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.
4. F. Gécseg and M. Steinby. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.
5. D. Giammarresi and A. Restivo. Two-dimensional languages. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, chapter 4, pages 215–267. Springer-Verlag, 1997.
6. A. Gupta. *Inductive Boolean Function Manipulation: A Hardware Verification Methodology for Automatic Induction.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1994.
7. A. Gupta and A. Fisher. Parametric circuit representation using inductive boolean functions. In *CAV 93*, volume 697 of *LNCS*, pages 15–28, 1993.
8. A. Gupta and A. Fisher. Representation and symbolic manipulation of linearly inductive boolean functions. In *Proc. of the IEEE Internation Conference on Computer-Aided Design*, pages 192–199. IEEE Computer Society, 1993.
9. A. Gupta and A. Fisher. Tradeoffs in canonical sequential representations. In *Proc. of the International Conference on Computer Design*, pages 111–116, 1994.
10. J. Henriksen, J. Jensen, M. Jorgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *TACS 95*, volume 1019 of *LNCS*, pages 89–110, 1996.
11. J. Hopcroft and J. Ullman. *Formal Languages and their Relation to Automata.* Addison-Wesley, 1969.
12. T. Jiang and B. Ravikumar. A note on the space complexity of some decision problems for finite automata. *IPL*, 40(1):25–31, 1991.
13. F. Klaedtke. *Induktive boolesche Funktionen, endliche Automaten und monadische Logik zweiter Stufe.* Master's thesis, Institut für Informatik, Albert-Ludwigs-Universität, Freiburg i. Br., 2000. in German.
14. N. Klarlund. Mona & Fido: The logic-automaton connection in practice. In *CSL 97*, volume 1414 of *LNCS*, pages 311–326, 1998.
15. G. Slutzki. Alternating tree automata. *TCS*, 41(2-3):305–318, 1985.
16. F. Somenzi. *CUDD: CU Decision Diagram Package, Release 2.3.0.* Department of Electrical and Computer Engineering, University of Colorado at Boulder, 1998.
17. M. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency*, volume 1043 of *LNCS*, pages 238–266, 1996.