Open Implication*

Karin Greimel[1], Roderick Bloem[1], Barbara Jobstmann[2], and Moshe Vardi[3]

[1]Graz University of Technology [2]EPFL [3]Rice University

Abstract. We argue that the usual trace-based notions of implication and equivalence for linear temporal logics are too strong and should be complemented by the weaker notions of open implication and open equivalence. Although open implication is harder to compute, it can be used to advantage both in model checking and in synthesis. We study the difference between trace-based equivalence and open equivalence and describe an algorithm to compute open implication of Linear Temporal Logic formulas with an asymptotically optimal complexity. We also show how to compute open implication while avoiding Safra's construction. We have implemented an open-implication solver for Generalized Reactivity(1) specifications. In a case study, we show that open equivalence can be used to justify the use of an alternative specification that allows us to synthesize much smaller systems in far less time.

1 Introduction

A recent verification project at STMicroelectronics [17] considered an arbiter that receives requests and provides acknowledgments. Two of the requirements for the design read: (R_1) From some time on, the difference between the total number of requests and the total number of acknowledgments is zero, and (R_2) the total number of acknowledgments never exceeds the total number of requests. Requirement R_1 does not imply R_2: a trace that contains an acknowledgment followed by a request with no further acknowledgments or requests thereafter fulfills R_1 but not R_2. Nevertheless, because one can not predict the number of requests that will come, the only way to implement R_1 is to always wait for a request before sending an acknowledge. Thus, any implementation that fulfills R_1 also fulfills R_2. We say that R_1 *open-implies* R_2. Thus, it suffices to make sure that R_1 holds; R_2 follows. Likewise, we say that two specifications are *open equivalent* if they are fulfilled by the same implementations.

Traditionally, for linear specification formalisms such as Linear Temporal Logic (LTL) [20] or Büchi automata [5], only trace implication and trace equivalence have been studied. Intuitively, trace implication and trace equivalence are defined with respect to all systems. In contrast, open implication and open equivalence are defined with respect to open systems only. In open systems we

* This work was supported by EU grant 217069 (COCONUT), the Swiss National Science Foundation (Indo-Swiss Research Program and NCCR MICS), NSF grants CCF-0613889, ANI-0216467, and CCF-0728882, BSF grant 9800096, and a gift from Intel. This paper is based on the MS thesis of the first author [8].

distinguish between inputs and outputs and we require that the system be *receptive* to all inputs [9], the intuition being that the system cannot block the actions of the environments.

The notions of open implication and open equivalence have not been studied in the literature. We argue here that these are important notions. When model checking open systems, a specification can always be substituted by an open-equivalent one: it is fulfilled by the same open systems. Likewise, for automatic synthesis of open systems from specifications [21], one may replace the specification by any realizable specification that open-implies it. The stronger specification may be easier to synthesize. Consider for instance, a simplified specification of an arbiter with input r for request and output a for acknowledgement. The specification reads $\varphi = (\mathsf{G}\,\mathsf{F}\,r) \rightarrow \mathsf{G}(a \rightarrow \mathsf{X}(\neg a\,\mathsf{U}\,r))$. Now consider $\varphi' = \mathsf{G}(a \rightarrow \mathsf{X}(\neg a\,\mathsf{W}\,r))$. We have that φ and φ' are open equivalent but not trace equivalent. Moreover, the language of φ' can be represented by a weak automaton and is thus both easier to model check [4, 15] and (much) easier to synthesize [10, 16].

In this paper, we show that the inability to predict the future is the underlying cause for the difference between open implication and trace implication. Then, we consider the problem of deciding whether φ open-implies ψ for LTL formulas φ and ψ. We provide an algorithm that runs in 2EXPTIME in $|\varphi|$ and PSPACE in $|\psi|$, matching the lower bounds. This algorithm uses Safra's intricate determinization construction. We complement this with an algorithm that avoids Safra's construction, is much easier to implement, and may be far more efficient when the specifications are not equivalent. Additionally, we consider *Generalized Reactivity(1)* formulas [19]. Although less expressive than LTL, such formulas suffice to conveniently describe most properties that occur in practice. Efficient synthesis tools for this subset have been used on realistic examples [2, 3]. We present an implementation of open implication based on this approach and show that it can be used to significantly simplify the synthesis of an arbiter for an industrial bus.

2 Preliminaries

We consider systems with input signals I and output signals O. We define $AP = I \cup O$, which is the set of atomic propositions in the logic specifications defined below. Our input alphabet is thus $D = 2^I$, the output alphabet is $\Sigma = 2^O$, and we define $\mathcal{A} = 2^{AP}$.

Transducers and Trees. We use transducers to represent open systems. A *(possibly infinite) transducer* with inputs D and outputs Σ is a tuple $T = (Q, q_0, \delta, \lambda)$, where Q is the (possibly infinite) state space, $q_0 \in Q$ is the initial state, $\delta : Q \times D \rightarrow Q$ is the transition function, and $\lambda : Q \rightarrow \Sigma$ is the output function. In each state, the transducer outputs a letter in Σ, then reads a letters in D, and moves to the next state. Transducers correspond to Moore machines. A transducer is *finite* if Q is finite. The *run* of T on a sequence $d = d_0 d_1 \cdots \in D^\omega$ is a

sequence $\rho_0\rho_1 \cdots \in Q^\omega$, where $\rho_0 = q_0$ and $\rho_{i+1} = \delta(\rho_i, d_i)$. The corresponding *word* is $\lambda(\rho) = w_0 w_1 \cdots \in \mathcal{A}^\omega$ such that $w_i = \lambda(\rho_i) \cup d_i$. The set $L(T)$ denotes the words corresponding to some run of T and is called the *language of* T.

We use *trees* to represent transducers and runs of alternating automata (below). A Σ-labeled D-tree is a tuple (τ, λ), where τ is the set of nodes, a prefix closed subset of D^*, and $\lambda : \tau \to \Sigma$ is the labeling function. If $\tau = D^*$, τ is *complete*. The node ε is the *root* of the tree and a node $t \cdot d$ is a *successor* of t. A *path* π in τ is a maximal sequence of nodes $t_0 t_1 \ldots$ such that $t_0 = \varepsilon$ and there are $d_0 d_1 \ldots$ such that $t_{i+1} = t_i \cdot d_i$. Paths can be finite or infinite. We assign to each path $\pi = t_0 t_1 \ldots$ a word $\lambda(\pi) = w_0 w_1 \ldots$ such that $w_i = \lambda(t_i) \cup d_i$ for all $i \geq 0$.

The *unrolling* of a transducer $T = (Q, q_0, \delta, \lambda)$ is a complete Σ-labeled D-tree (τ, λ), such that each run ρ of T is mapped to an infinite path π in (τ, λ) with $\lambda(\rho) = \lambda(\pi)$. A tree is *regular* if it is the unrolling of some finite transducer. We denote the set of all regular Σ-labeled trees with directions D by $\mathcal{T}$.

Temporal Logics. We write specifications in *Linear Temporal Logic (LTL)* [20]. The syntax of LTL is defined in negation normal form as $\varphi ::= \mathsf{true} \mid \mathsf{false} \mid p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\varphi \mid \varphi \,\mathsf{U}\, \varphi \mid \varphi \,\mathsf{R}\, \varphi$ with $p \in AP$. We use the usual semantics of LTL for words in $\mathcal{A}^\omega$. The set of words that satisfies φ is denoted by $L(\varphi) \subseteq \mathcal{A}^\omega$. A Σ-labeled D-tree t satisfies φ if for all paths π of t, $\pi \models \varphi$. A transducer T satisfies φ ($T \models \varphi$) if its unrolling does. A formula φ is *satisfiable* if $L(\varphi) \neq \emptyset$, it is *tautologous* if $L(\varphi) = \mathcal{A}^\omega$ and it is *realizable* if there is a tree t such that $t \models \varphi$.

Automata. Let $\mathbb{B}^+(X)$ denote the set of Boolean formulas without negations over X. We say that a set $C \subseteq 2^X$ *satisfies* $\varphi \in \mathbb{B}^+(X)$ (written $C \models \varphi$) if φ evaluates to true after replacing all occurrences of $c \in C$ ($c \notin C$) in φ by true (false, resp.). Set C is *minimal* if forall $c \in C$, $(C \setminus \{c\}) \not\models \varphi$.

An *alternating parity tree automaton* for Σ-labeled D-trees is a tuple $A = (Q, q_0, \delta, F)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to \mathbb{B}^+(Q \times D)$ is the transition relation and the acceptance condition $F = (F_1, \ldots, F_k)$ is a partition of Q, where k is the *index* of A. We use A^q to denote the automaton A with initial state q.

We say that an alternating tree automaton is *nondeterministic* if it does not force multiple copies to one child. That is, for all $q \in Q$ and $\sigma \in \Sigma$, if $C \models \delta(q, \sigma)$ and C is minimal then for all $(q, d) \in C$ and $(q', d') \in C$, if $d = d'$ then $q = q'$. The automaton is *universal* if all formulas are conjunctions and it is *deterministic* if it is both nondeterministic and universal. For deterministic automata we can assume, without loss of generality, that the transition relation is of the form $\delta : Q \times \Sigma \times D \rightharpoonup Q$. An automaton is a *co-Büchi* automaton if $k = 2$ and a *Büchi* automaton if $k = 3$ and $F_1 = \emptyset$. Tree automata run on trees with directions D. If $|D| = 1$, we say the automaton runs on *words* (over Σ) and omit D.

A *run* of an alternating tree automaton A on a tree (τ_I, λ_I) is a tree $T_\rho = (\tau_\rho, \lambda_\rho)$ with $\tau_\rho \subseteq \mathbb{N}^*$ and $\lambda_\rho : \tau_\rho \to (Q \times \tau_I)$ for which (1) $\lambda_\rho(\varepsilon) = (q_0, \varepsilon)$ and (2) If t_ρ is a node of T_ρ with label (q, t_I) and the children of t_ρ are labeled

$(q_1, t_1), \ldots, (q_n, t_n)$, then for all $i \in \{1, \ldots, n\}$ there is a $d_i \in D$ such that $t_i = t_I \cdot d_i$ and $\{(q_1, d_1), \ldots, (q_n, d_n)\} \models \delta(q, \lambda_I(t_I))$. (Not all directions must appear in $\{(q_1, d_1), \ldots, (q_n, d_n)\}$.) Let $\pi = t_0 t_1 \ldots$ be an infinite path in $(\tau_\rho, \lambda_\rho)$, then $\inf(\pi) = \{q \in Q \mid$ there exist infinitely many nodes $t \in \pi$ with $\lambda_\rho(t) = (q, t_I)$ for some $t_I\}$. A path is *accepting* if the minimal $i \in \{1, \ldots, k\}$ for which $\inf(\pi) \cap F_i \neq \emptyset$ is even. A run is accepting if all infinite paths are accepting. An automaton accepts an input tree (τ_I, λ_I), if there exists an accepting run on (τ_I, λ_I). We call the set of trees accepted by A the *language of* A and denote it by $L(A)$.

We use three letter acronyms for automata, where the first denotes the branching mode of the automaton (nondeterministic, universal, deterministic, or alternating), the second describes the acceptance condition (parity, Büchi or co-Büchi), and the third letter indicates the input elements (words or trees). For instance, a UPT is a universal parity tree automaton.

3 Open Implication

3.1 Definitions, Characteristics, and Lower Bounds

Definitions. Let us first recall the standard notions of implication and equivalence between two LTL formulas and then define open implication and open equivalence.

Definition 1. *Given two LTL formulas φ and ψ, φ* trace-implies *ψ if $L(\varphi) \subseteq L(\psi)$. Formula φ is* trace equivalent *to ψ if $L(\varphi) = L(\psi)$.*

Definition 2. *Given two LTL formulas φ and ψ, φ* open-implies *ψ, denoted by $\varphi \multimap \psi$, if for all (infinite) transducers T we have that $T \models \varphi$ implies $T \models \psi$. Likewise, $\varphi \multimap\!\!\circ\!\!\multimap \psi$ (φ is* open equivalent *to ψ) if $\varphi \multimap \psi$ and $\psi \multimap \varphi$.*

Theorem 1. *If for all finite transducers T, $T \models \varphi$ implies $T \models \psi$, then $\varphi \multimap \psi$.*

Proof. We prove the converse. If $\varphi \not\multimap \psi$, then there is a (possibly infinite) transducer T such that $T \models \varphi$, but $T \not\models \psi$. The unrolling of T is accepted by the deterministic Streett automaton A that accepts all trees (of the proper arity) satisfying the CTL* formula $\chi = \mathsf{A}\varphi \wedge \neg \mathsf{A}\psi$ [7]. Since the language of A is not empty, there exists a finite transducer generating a tree accepted by A [22]. Thus, there exists a finite transducer T' such that $T' \models \varphi$, but $T' \not\models \psi$.

Without loss of generality, we refer to finite transducers in the remainder of the paper.

Open versus Trace Equivalence. If two specifications are open equivalent but not trace equivalent, then the traces in which the specifications differ cannot be produced by a transducer because they require knowledge of the future.

Rosner [23] distinguishes two reasons for unrealizability. First, a specification may be unrealizable because there is an infinite input word that cannot be paired

with an output word. The second reason is that some specifications require clairvoyance. For instance, for the specification $a \leftrightarrow \mathsf{X}\, r$, where a is an output and r is an input, there exists a valid output word for every input word. Lack of knowledge of the future input prevents an implementation. (See also [29].)

Formally, given a specification φ, we call $w \in \mathcal{A}^\omega$ *φ-clairvoyant* if $w \models \varphi$ but for some prefix $w' \cdot (i \cup o)$ there is no transducer T that outputs o in the initial state, such that for all words v of T, $w' \cdot v \models \varphi$. That is, the word cannot be used in a transducer because after some point, there is no correct reaction to all future inputs. Note that only clairvoyant words satisfy $a \leftrightarrow \mathsf{X}\, r$. A word that is not φ-clairvoyant is called *φ-secure*.

If two realizable specifications φ and ψ are open equivalent, then the set of φ-secure words and the set of ψ-secure words are equal.

Theorem 2. *We have $\varphi \multimap\!\!\!\!\circ\, \psi$ iff $L(\varphi) \setminus L(\psi)$ consists of φ-clairvoyant words.*

Proof. The key insight is that w is φ-secure iff there is a transducer T such that $T \models \varphi$ and $w \in L(T)$.

Let $w \in L(\varphi) \setminus L(\psi)$. If w is φ-secure then there is a transducer that satisfies φ, contains w, and thus does not satisfy ψ, so $\varphi \not\multimap\!\!\!\!\circ\, \psi$. Vice-versa, suppose that $\varphi \not\multimap\!\!\!\!\circ\, \psi$. Then there is a transducer T that satisfies φ and not ψ. This transducer contains a word w that satisfies φ and not ψ and this word is φ-secure.

Extending our notation to ω-regular languages, we have that for every ω-regular language L there is an ω-regular language L' that consists of the L-secure words in L. Language L' is the unique minimal representative of the open-equivalence class of L and precisely characterizes all transducers that satisfy L. The language can be constructed from a DPW A with language L by removing all edges $(q, o \cup i, q')$ such that there is an i' with $(q, o \cup i', q'') \in \delta$ and $L(A^{q''})$ is not realizable.

Lower Bounds. An obvious solution to deciding open implication is to apply the approach suggested in the proof of Theorem 1 by checking nonemptiness of the tree automaton for the formula χ. The problem with this algorithm is that it is doubly exponential in both $|\varphi|$ and $|\psi|$. After discussing the lower-bound complexity of deciding open equivalence and open implication between two LTL formulas, we describe an asymptotically-optimal algorithm.

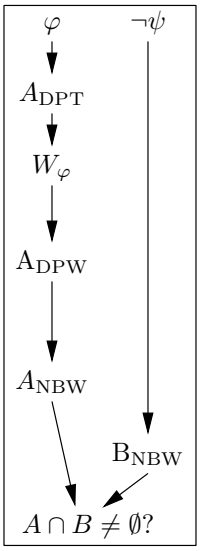
Theorem 3. *Let φ and ψ be two LTL formulas. (1) Deciding whether $\varphi \multimap\!\!\!\!\circ\, \psi$ is 2EXPTIME-hard, so is deciding whether $\varphi \multimap\!\!\!\!\circ\!\!\!\!\circ\, \psi$ and (2) Deciding whether $\varphi \multimap\!\!\!\!\circ\, \psi$ is 2EXPTIME-hard for a fixed ψ and PSPACE-hard for a fixed φ.*

Proof. We have that φ is unrealizable iff $\varphi \multimap\!\!\!\!\circ\, \mathsf{false}$ iff $\varphi \multimap\!\!\!\!\circ\!\!\!\!\circ\, \mathsf{false}$ and LTL-realizability is 2EXPTIME-complete [21]. This proves 2EXPTIME-hardness.

We prove that ψ is tautologous iff $\mathsf{true} \multimap\!\!\!\!\circ\, \psi$. Because deciding validity of LTL formulas is PSPACE-complete [25], this proves that open implication is PSPACE-hard in ψ. The forward direction is trivial. For the other direction, assume that ψ is not tautologous, then $\exists w \in \mathcal{A}^\omega : w \not\models \psi$. Since we can choose w as a finite prefix followed by a finite cycle [27], we can construct a transducer T such that w is a word of T. We have that $T \models \mathsf{true}$ but $T \not\models \psi$, so $\mathsf{true} \not\multimap\!\!\!\!\circ\, \psi$.

3.2 Algorithm and Upper Bounds

We show an algorithm to decide whether $\varphi \twoheadrightarrow \psi$ that runs in time doubly exponential in φ and in space polynomial in ψ. We first describe an algorithm that is exponential in ψ, and then show how to obtain optimal space complexity. In the following, we fix $n = |\varphi|$ and $m = |\psi|$.



The algorithm proceeds as follows:

1. Construct a DPT $\mathrm{A}_{\mathrm{DPT}} = (Q_{\mathrm{DPT}}, q_{0\mathrm{DPT}}, \delta_{\mathrm{DPT}}, F_{\mathrm{DPT}})$ such that $L(\mathrm{A}_{\mathrm{DPT}}) = \{t \in \mathcal{T} \mid t \models \varphi\}$ with at most $2^{n2^{2n+2}+4n}$ states and index $i_{\mathrm{DPT}} = 2^{2n+1}$ [18, 27].
2. Compute the set $W_\varphi = \{q \in Q_{\mathrm{DPT}} \mid L(\mathrm{A}_{\mathrm{DPT}}{}^q) \neq \emptyset\}$ in doubly exponential time in n [7].
3. Construct a DPW $\mathrm{A}_{\mathrm{DPW}} = (Q_{\mathrm{DPW}}, q_{0\mathrm{DPW}}, \delta_{\mathrm{DPW}}, F_{\mathrm{DPW}})$ over AP with $|Q_{\mathrm{DPT}}|$ states and index i_{DPT} such that $\sigma \in L(\mathrm{A}_{\mathrm{DPW}})$ iff $\sigma = \lambda(\pi)$ for some path π of a tree $t \in L(\mathrm{A}_{\mathrm{DPT}})$ (see below).
4. Construct a NBW $\mathrm{A}_{\mathrm{NBW}} = (Q_{\mathrm{NBW}}, q_{0\mathrm{NBW}}, \delta_{\mathrm{NBW}}, F_{\mathrm{NBW}})$ with at most $2|Q_{\mathrm{DPT}}|i_{\mathrm{DPT}}$ states, such that $L(\mathrm{A}_{\mathrm{NBW}}) = L(\mathrm{A}_{\mathrm{DPW}})$ [12].
5. Construct an NBW $\mathrm{B}_{\mathrm{NBW}}$ with at most 2^{2m} states that accepts all words in $L(\neg\psi)$ [27].
6. Check if $L(\mathrm{A}_{\mathrm{NBW}}) \cap L(\mathrm{B}_{\mathrm{NBW}}) = \emptyset$ in time linear in the size of $\mathrm{A}_{\mathrm{NBW}}$ and $\mathrm{B}_{\mathrm{NBW}}$ [27].

The DPW $\mathrm{A}_{\mathrm{DPW}} = (Q_{\mathrm{DPW}}, q_{0\mathrm{DPT}}, \delta_{\mathrm{DPW}}, F_{\mathrm{DPW}})$ is constructed as follows. We have $Q_{\mathrm{DPW}} = W_\varphi$, $\delta_{\mathrm{DPW}}(q, o \cup i) = \delta_{\mathrm{DPT}}(q, o, i)$ if $\forall j \in I : \delta_{\mathrm{DPT}}(q, o, j) \in W_\varphi$, and F_{DPW} equals F_{DPT} restricted to states in Q_{DPW}.

Lemma 1. *$\sigma \in L(\mathrm{A}_{\mathrm{DPW}})$ iff $\exists t \in L(\mathrm{A}_{\mathrm{DPT}})$ with a path π such that $\lambda(\pi) = \sigma$.*

Proof. Let $\sigma = \sigma_0\sigma_1 \cdots \in L(\mathrm{A}_{\mathrm{DPW}})$ and suppose that $\sigma_j = i_j \cup o_j$ with $i_j \in D$ and $o_j \in \Sigma$. Then $\forall i \in D$, the run of the DPT for $\sigma_0\sigma_1 \ldots \sigma_{j-1}(o_j \cup i)$ ends in a state in W_φ, whence we can extend the path to an accepted tree that includes the word σ. Vice versa, if there exists a tree $t \in L(\mathrm{A}_{\mathrm{DPT}})$ with a path π such that $\lambda(\pi) = \sigma$ then, by construction, σ is accepted by $\mathrm{A}_{\mathrm{DPW}}$.

Theorem 4. *$\varphi \not\twoheadrightarrow \psi$ iff $L(\mathrm{A}_{\mathrm{NBW}}) \cap L(\mathrm{B}_{\mathrm{NBW}}) \neq \emptyset$.*

Proof. If $\varphi \not\twoheadrightarrow \psi$ then there is a transducer T such that $T \models \varphi$ and $T \not\models \psi$, so for some path $\pi \in t$ where t is the unrolling of T, we have $\lambda(\pi) \not\models \psi$ and $\lambda(\pi) \models \varphi$. Thus, $\lambda(\pi)$ is in $L(\mathrm{A}_{\mathrm{NBW}}) \cap L(\mathrm{B}_{\mathrm{NBW}})$. Similarly, if there is a word σ in $L(\mathrm{A}_{\mathrm{NBW}}) \cap L(\mathrm{B}_{\mathrm{NBW}})$ then there is a regular tree $t \in \mathcal{T}$ satisfying φ with a path $\pi \in t$ such that $\lambda(\pi) = \sigma$. The transducer T generating t models φ and violates ψ (because $\sigma \not\models \psi$), so $\varphi \not\twoheadrightarrow \psi$ holds.

Theorem 5. *Deciding $\varphi \twoheadrightarrow \psi$ is 2EXPTIME-complete and PSPACE-complete when φ is fixed. Deciding open equivalence is 2EXPTIME-complete.*

Proof. Hardness was shown in Theorem 3. The algorithm runs in time $2^{2^{O(n)}}2^{O(m)}$. The first four steps of the algorithm use time and space $2^{2^{O(n)}}$.

Deciding whether $L(\mathrm{A_{NBW}}) \cap L(\mathrm{B_{NBW}}) = \emptyset$ can be done within the resources required. The key is avoiding an explicit construction of $\mathrm{B_{NBW}}$, rather, constructing its state while performing an on-the-fly search. We check whether there is a word that is accepted by the NBW $\mathrm{A_{NBW}}$ and the NBW $\mathrm{B_{NBW}}$ by nondeterministically guessing a word $\sigma \in \mathcal{A}^\omega$ and simultaneously keeping track of the corresponding runs in both automata. We only have to store two states of the NBW $\mathrm{A_{NBW}}$ and two states of the NBW $\mathrm{B_{NBW}}$ at each step of the algorithm. Since each state of $\mathrm{A_{NBW}}$ has size $2^{O(n)}$ and each state of $\mathrm{B_{NBW}}$ has size $O(m)$, nonemptiness for $L(\mathrm{A_{NBW}}) \cap L(\mathrm{B_{NBW}})$ can be checked using $2^{O(n)} + O(m)$ nondeterministic space. By [24], this can be done using $2^{O(n)} + O(m^2)$ deterministic space. The time requirement is exponential in the space requirement, so it is $2^{2^{O(n)}}2^{O(m^2)}$.

Altogether, the algorithm uses doubly exponential time in n and polynomial space in m.

Open implication can be viewed as a simultaneous realizability testing for the implicate (left-hand-side of implication) and validity testing for the implicant (right-hand-side of the implication). For a fixed implicant, open implication is 2EXPTIME-complete, just like realizability[1], and for a fixed implicate open implication is PSPACE-complete, just like validity.

For the next section, we need a bound on the size of the witness for $\varphi \not\twoheadrightarrow \psi$.

Lemma 2. *If $L(\mathrm{A_{NBW}}) \cap L(\mathrm{B_{NBW}}) \neq \emptyset$ then there exists a word $uv \in \mathcal{A}^*$ of length at most $2^{2m+1}|Q_{\mathrm{NBW}}|$ such that $uv^\omega \in L(\mathrm{A_{NBW}}) \cap L(\mathrm{B_{NBW}})$.*

Proof. The product automaton $\mathrm{C_{NBW}}$ of $\mathrm{A_{NBW}}$ and $\mathrm{B_{NBW}}$ has at most $2 \cdot 2^{2m}|Q_{\mathrm{NBW}}|$ states. If $L(\mathrm{C_{NBW}}) \neq \emptyset$ then there exists a word $uv \in \mathcal{A}^*$ whose length is at most the number of states in $\mathrm{C_{NBW}}$ such that $uv^\omega \in L(\mathrm{C_{NBW}})$.

Theorem 6. *If $\varphi \not\twoheadrightarrow \psi$, there is a transducer T with at most $2^{2m+1}|Q_{\mathrm{NBW}}||Q_{\mathrm{DPT}}|$ states such that $T \models \varphi$ but $T \not\models \psi$.*

Proof. Let $\pi = (i_0 \cup o_0) \dots (i_{k-1} \cup o_{k-1})\big((i_k \cup o_k) \dots (i_{l-1} \cup o_{l-1})\big)^\omega$ in $L(\mathrm{A_{NBW}}) \cap L(\mathrm{B_{NBW}})$ with $l \leq 2^{2m+1}|Q_{\mathrm{NBW}}|$. The transducer is $T = (Q, q_0, \delta, \lambda)$ with $Q = W_\varphi \times \{0, \dots, l-1, \bot\}$, where the second element keeps track of whether and where we are in π. From $\mathrm{A_{DPT}}$ we can derive a transducer T' with state space W_φ that satisfies φ. Our transducer T behaves like T' for all states in $Q_{\mathrm{DPT}} \times \{\bot\}$. For $j \in \{0, \dots, l-1\}$, we have $\lambda((q,j)) = o_j$ and $\delta((q,j), i) = (\delta_{\mathrm{DPT}}(q, o_j, i), j')$, where $j' = \bot$ if $i \neq i_j$ and if $i = i_j$ then $j' = j+1$ if $j < l-1$ and k otherwise. Note that $\delta_{\mathrm{DPT}}(q, o_j, i) \in W_\varphi$ because π is accepted by $\mathrm{A_{NBW}}$ and thus by $\mathrm{A_{DPW}}$. The transducer violates ψ when the input sequence is as in π and satisfies φ. The number of states of T is at most $2^{2m+1}|Q_{\mathrm{NBW}}||Q_{\mathrm{DPT}}| = 2^{n2^{2n+3}+10n+3+2m}$

[1] In spite of the doubly exponential lower bound, there have been recently encouraging developments regarding the practicality of realizability checking [10, 16, 19].

Notation: Let $witn(n,m) = 2^{n2^{2n+3}+10n+3+2m}$.

Note that it is possible to avoid constructing $\mathrm{A}_{\mathrm{NBW}}$ in our algorithm, if we check language emptiness of $L(\mathrm{A}_{\mathrm{DPW}}) \cap L(\mathrm{B}_{\mathrm{NBW}})$ directly. This leads to a slightly better upper bound in Theorem 6.

Our proof techniques can be extended to other linear specification formalisms that allow a translation of the specification into an NBW. Two popular formalisms falling into that class are QPTL [26] and the industrial PSL [6]. The algorithm follows the approach described above, adapting Step 1 and 5 to the formalism used. The complexity of the algorithm depends on the cost of translating the specification into an NBW. For QPTL and PSL it is possible to find an algorithm whose complexity matches the lower bounds for realizability and validity of the respective logics.

Note that the use of quantifiers allows us to check open equivalence between specifications at different levels of abstraction, e.g., a specification can be checked against a refined version that includes variables encoding implementation details. This is particular useful for synthesis of Generalized Reactivity(1) (cf. Section 4), which introduces additional variables to encode LTL specifications.

3.3 Avoiding Safra's Construction

In this section, we present another algorithm to decide if $\varphi \twoheadrightarrow \psi$, based on [16], which avoids Safra's intricate determinization construction and parity games and lends itself to implementation [10].

In [16], Kupferman and Vardi provide an approach to decide the realizability problem for LTL. Given an LTL formula φ, they construct a UCT U that accepts exactly all trees that are solutions to the realizability problem of φ.

Theorem 7. [16] *The realizability problem for an LTL formula φ can be reduced to the nonemptiness problem for a UCT with at most $4^{|\varphi|}$ states.*

In order to check if the language of U is empty, U is translated into a corresponding NBT N.

Theorem 8. [16] *Let U be a UCT with p states. For each $k > 0$ we can construct an NBT N_k with $2^{p(\log(k)+2)}$ states such that a tree generated by a transducer with at most k states is accepted by U iff it is accepted by N_k.*

Intuitively, the size of N_k is bounded by the size of the transducers generating the trees N_k has to accept. (Note that in general one cannot translate a UCT to an equivalent NBT.)

Since we are looking for a transducer that fulfills φ and violates ψ, Theorem 6 provides a bound on the size of the transducers of interest, which is $witn(n,m)$. We can replace the algorithm of Section 3.2 by the following algorithm:

1. Construct a UCT $\mathrm{A}_{\mathrm{UCT}}$ of size 4^n such that $L(\mathrm{A}_{\mathrm{UCT}}) = \{t \in \mathcal{T} \mid t \models \varphi\}$. From $\mathrm{A}_{\mathrm{UCT}}$ construct the NBT $N_{witn(n,m)}$ (Theorem 8). The number of states of this NBT is $2^{O(m)2^{O(n)}}$.

2. Compute the set W_φ of states q of $N_{witn(n,m)}$, such that $N^q_{witn(n,m)}$ accepts some tree, in quadratic time [28].
3. From $N_{witn(n,m)}$ construct an NBW $\mathrm{A_{NBW}}$ such that $\sigma \in L(\mathrm{A_{NBW}})$ if $\sigma = \lambda(\pi)$ for some path π of a tree $t \in L(N_{witn(n,m)})$.
4. Construct an NBW $\mathrm{B_{NBW}}$ with at most 4^m states that accepts all words in $L(\neg\psi)$ [27].
5. Check if $L(\mathrm{A_{NBW}}) \cap L(\mathrm{B_{NBW}}) = \emptyset$ in time linear in the size of $\mathrm{A_{NBW}}$ and $\mathrm{B_{NBW}}$ [27].

Theorem 9. *Deciding if $\varphi \twoheadrightarrow \psi$ can be reduced to the language emptiness check of the product between $\mathrm{A_{NBW}}$ and $\mathrm{B_{NBW}}$.*

The revised algorithm is doubly exponential in φ and exponential in ψ. We do not attempt to be space efficient here, as the automaton $N_{witn(n,m)}$ is already exponential in ψ. Nevertheless, this approach is useful as it avoids Safra's construction and parity games. It is particularly suitable for finding counterexamples to open implication, since it can be implemented incrementally by increasing the size of the transducers we are looking for. This may allow us to find counterexamples using much smaller automata than the full deterministic parity automaton [16, 14].

4 Generalized Reactivity

Generalized Reactivity(1), or GR(1) for short, is a specification formalism that has been proposed in [19] for synthesis. GR(1) specifications consist of two sets of symbolically represented DBWs, one for the environment and one for the system. This formalism avoids the determinization step normally required for synthesis; it has a symbolic synthesis algorithm consisting of a triply nested fixpoint computation [19]. Experience shows that the formalism can be used to synthesize modest sized industrial circuits from their specifications, and that the restriction to GR(1) specifications is not overly restrictive [2, 3].

We briefly recapitulate the construction of [19]. A DBW over AP with n states can be symbolically represented by an LTL formula φ by using a set V of $\lceil \lg(n) \rceil$ new atomic propositions. The formula is a conjunction of three parts: (1) φ^i is a propositional formula over V denoting the initial state, (2) φ^t is a formula of the form $\mathsf{G} \bigwedge_i (\chi_i \rightarrow \mathsf{X}\,\xi_i)$ representing the complete, deterministic transition relation, where χ_i and ξ_i are propositional formulas over $AP \cup V$ and V, respectively, and (3) φ^f is a formula of the form $\mathsf{G}\,\mathsf{F}\,\chi$, where χ is a propositional formula over V, representing the fairness condition. For instance, we represent $\mathsf{G}(r \rightarrow \mathsf{F}\,a)$ with $V = \{s\}$ by $\psi = \neg s \wedge \mathsf{G}\big((\neg s \wedge r \wedge \neg a \rightarrow \mathsf{X}\,s) \wedge (\neg s \wedge (\neg r \vee a) \rightarrow \mathsf{X}\neg s) \wedge (s \wedge \neg a \rightarrow \mathsf{X}\,s) \wedge (s \wedge a \rightarrow \mathsf{X}\neg s)\big) \wedge \mathsf{G}\,\mathsf{F}\neg s$. The DBW is shown in Figure 1.

A GR(1) specification has the form $\varphi = (\bigwedge_j \varphi_{e,j}) \rightarrow (\bigwedge_j \varphi_{s,j})$, where environment assumptions $\varphi_{e,j}$ and the system guarantees $\varphi_{s,j}$ represent DBWs. In the sequel, let $\varphi^a_b = \bigwedge_j \varphi^a_{b,j}$ for $a \in \{i, t, f\}$ and $b \in \{s, e\}$. GR(1) formulas are intended to describe Mealy machines, not Moore machines, which leads to small technical differences with the previous presentation. Also, in keeping with [19],

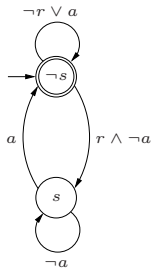


Fig. 1. DBW for $\mathsf{G}(r \rightarrow \mathsf{F}\, a)$

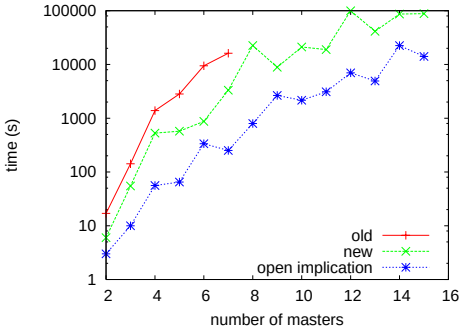


Fig. 2. Time needed for calculations

we use game-based terminology here. A *game* corresponds to a tree automaton and a *winning state* corresponds to a state with a nonempty language.

In order to decide realizability of a GR(1) formula, a game graph G_φ is built. The transition structure of the game graph is given by the combination of φ_e^t and φ_s^t, the initial state is $\varphi_e^i \wedge \varphi_s^i$, and the winning condition is $\varphi_e^f \rightarrow \varphi_s^f$. The winning region W_φ of the game is computed symbolically by a triply nested fixpoint formula, and the formula is realizable if the initial state is winning [19].

We now describe how to decide open implication. Suppose we have two GR(1) specifications, $\varphi = \bigwedge_i \varphi_{e,i} \rightarrow \bigwedge_j \varphi_{s,j}$ and $\psi = \bigwedge_k \psi_{e,k} \rightarrow \bigwedge_l \psi_{s,l}$. We check whether $\varphi \twoheadrightarrow \psi$ as follows.

1. Construct the game graph G_φ and compute the winning region W_φ.
2. Construct the game graph G_ψ and the product G of G_φ and G_ψ.
3. Check if there is a path in G that (i) stays within W_φ, (ii) satisfies $\varphi_e^f \rightarrow \varphi_s^f$, and (iii) violates $\psi_e^f \rightarrow \psi_s^f$.

Note that this algorithm is similar to the one described above, although removal of the losing states (the states with an empty language) has been replaced by the requirement that the path remain in the set of winning states. Thus, we are looking for a path within W_φ that satisfies all of the $\psi_{e,k}^f$, violates one of the $\psi_{s,l}^f$, and either violates one if the $\varphi_{e,i}^f$ or satisfies all of the $\varphi_{s,j}^f$. This is expressed by the μ-calculus [13] formula $\gamma = \mu Y \,.\, W_\varphi \wedge (\gamma' \vee pre(Y))$, where $\gamma' = \bigvee_{i,l}(\nu Y \,.(W_\varphi \wedge \neg\varphi_{e,i}^f \wedge E_\psi \wedge \neg\psi_{s,l}^f)) \vee \bigvee_l(\nu Y \,.(W_\varphi \wedge S_\varphi \wedge E_\psi \wedge \neg\psi_{s,l}^f))$, $E_\psi = \bigwedge_k pre(\mu Z \,.\, Y \wedge (\psi_{e,k}^f \vee pre(Z)))$, and $S_\varphi = \bigwedge_j pre(\mu Z \,.\, Y \wedge (\varphi_{s,j}^f \vee pre(Z)))$.

The complexity of a symbolic algorithm can be given in terms of the number of symbolic steps [1], where steps in this case are preimage computations and computations of the force operator used for games [19]. Computing the winning region of G_φ requires a cubic number of steps in terms of the number of states in G_φ. Computing γ, a doubly-nested fixpoint, and thus requires only a quadratic number of steps in terms of the size of G.

Theorem 10. *We have that $\varphi \not\twoheadrightarrow \psi$ iff the initial state of G is in the set γ. This computation uses a number of symbolic steps cubic in G_φ and quadratic in G_ψ.*

4.1 Experimental Results

We have implemented the algorithm for open implication of GR(1) formulas in Anzu [11], a synthesis tool for GR(1) specifications. We have tested our implementation on specifications of an arbiter for ARM's AMBA AHB bus used in [2, 3]. In Figure 2, we show the time Anzu takes to synthesize the specifications and the time needed to calculate open implication. The *old* specification, which was used in [2] can only be synthesized for up to 7 masters. Anzu runs out of memory for larger instances. In [3] an improved version of the specification was presented, but it was not proven that the *old* and *new* specifications are equivalent. The *new* specification can be synthesized for up to 15 masters[2]. (2GB of memory were available.) Using the algorithm presented above, we can show that the *new* specification open-implies the *old* one and can thus be used in its stead. Figure 2 also shows that the combined time needed to calculate open implication and to synthesize the *new* specification is less than the time needed to synthesize the *old* specification, when that is possible. It should be noted that the circuits that result from the *new* specification are much smaller than those resulting from the *old* specification.

5 Conclusions

We have argued that open implication is an important concept both in model checking and in synthesis. We have given algorithms to compute open implication and open equivalence for the specification formalisms LTL and GR(1). For LTL, we have shown an algorithm that runs in time that is doubly exponential in the size of the implicate and space that is polynomial in the size of the implicant, matching the lower bounds. We have also shown how to implement the algorithm while avoiding Safra's construction. Finally, we implemented the approach for GR(1) specifications and showed that it can be used to show the correctness of simple specifications and, thus, to synthesize circuits that would otherwise be out of reach.

References

1. R. Bloem, H. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. *Formal Methods in System Design*, 2006.
2. R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Automatic hardware synthesis from specifications: A case study. In *DATE*, 2007.
3. R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Specify, compile, run: Hardware from PSL. In *6th International Workshop on Compiler Optimization Meets Compiler Verification*, pages 3–16, 2007.
4. R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *Computer Aided Verification (CAV'99)*. 1999.
5. J. R. Büchi. On a decision method in restricted second order arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, 1962.

[2] Our specifications are slightly different from those used in [2, 3]

6. C. Eisner and D. Fisman. *A Practical Introduction to PSL*. Springer, 2006.
7. E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *Proc. Foundations of Computer Science*, 1988.
8. K. Greimel. Open implication. Master's thesis, Graz University of Technology, 2007.
9. D. Harel and A. Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, pages 477–498. 1985.
10. B. Jobstmann and R. Bloem. Optimizations for LTL synthesis. In *6th Conference on Formal Methods in Computer Aided Design (FMCAD'06)*, pages 117–124, 2006.
11. B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem. Anzu: A tool for property synthesis. In *Computer Aided Verification*, pages 258–262, 2007.
12. V. King, O. Kupferman, and M. Y. Vardi. On the complexity of parity word automata. In *Foundations of Software Science and Computation Structures*, 2001.
13. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
14. O. Kupferman, N. Piterman, and M. Y. Vardi. Safraless compositional synthesis. In *Conference on Computer Aided Verification (CAV'06)*, pages 31–44, 2006.
15. O. Kupferman and M. Y. Vardi. Relating linear and branching model checking. In *IFIP Working Conference on Programming Concepts and Methods*, 1998.
16. O. Kupferman and M. Y. Vardi. Safraless decision procedures. In *Symposium on Foundations of Computer Science (FOCS'05)*, pages 531–542, 2005.
17. A. McIsaac, November 2006. Personal Communication.
18. N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Logic in Computer Science (LICS'06)*, pages 255–264, 2006.
19. N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In *Proc. Verification, Model Checking and Abstract Interpretation*, pages 364–380, 2006.
20. A. Pnueli. The temporal logic of programs. In *IEEE Symposium on Foundations of Computer Science*, pages 46–57, Providence, RI, 1977.
21. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. Symposium on Principles of Programming Languages (POPL '89)*, pages 179–190, 1989.
22. M. O. Rabin. *Automata on Infinite Objects and Church's Problem*. Regional Conference Series in Mathematics. American Mathematical Society, Providence, RI, 1972.
23. R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, 1992.
24. W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, April 1970.
25. A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logic. *Journal of the ACM*, 3:733–749, 1985.
26. A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 1987.
27. M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.
28. M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32(2):182–221, 1986.
29. N. Yoshiura. Finding the causes of unrealizability of reactive system formal specifications. In *Proc. Software Engineering and Formal Methods (SEFM'04)*, 2004.