Synthesis with Clairvoyance

Orna Kupferman[1], Dorsa Sadigh[2], and Sanjit A. Seshia[2]

[1] Hebrew University, School of Engineering and Computer Science, Jerusalem, Israel
[2] UC Berkeley, EECS department, Berkeley CA, USA

Abstract. We consider the problem of automatically synthesizing, from a linear temporal logic (LTL) specification, a system that is guaranteed to satisfy the specification with respect to all environments. Algorithms for solving the synthesis problem reduce it to the solution of a game played between the system and its environment, in which the system and environment alternate between generating outputs and inputs respectively. Typically, the system is required to generate an output right after receiving the current input. If a solution to the game exists, the specification is said to be realizable.

In this paper, we consider the role of clairvoyance in synthesis, in which the system can "look into the future," basing its output upon future inputs. An infinite look-ahead transforms the realizability problem into a problem known as universal satisfiability. A thesis we explore in this paper is that the notion of clairvoyance is useful as a heuristic even in the general case of synthesis, when there is no lookahead. Specifically, we suggest a heuristic in which we search for strategies where the system and the environment try to force each other into hopeless states in the game — states from which they cannot win, no matter how large the lookahead. The classification to hopeful and hopeless states is thus based on a modified notion of universal satisfiability where the output prefix is constrained. Our approach uses the automata for the specification in the process of classification into hopeful and hopeless states, and uses the structure of the automata in order to construct the game graph, but the important point is that the game itself is a reachability game. We demonstrate the efficiency of our approach with examples, and outline some directions for future work exploring the proposed approach.

1 Introduction

A frequent criticism against verification methods is that verification is done after significant resources have already been invested in the development of the system. The critics argue that the desired goal is to use the specification in the system development process in order to guarantee the design of correct systems. This is called *automatic synthesis*. Formally, given a specification to a reactive system, typically by means of an LTL formula, the goal in automatic synthesis is to transform it into a system that is guaranteed to satisfy the specification.[3]

[3] To make life interesting, several different methodologies in system design are all termed "synthesis". The automatic synthesis we study should not be confused with *logic synthesis*, which is a process by which an abstract form of a desired circuit behavior (typically, register transfer level, which by itself may be the outcome of yet another synthesis procedure, termed *high-level synthesis*) is turned into a design implementation by means of logic gates.

In the late 1980s, several researchers realized that the classical approach to system synthesis, where a system is extracted from a proof that the specification is satisfiable, is well suited to *closed* systems, but not to *open* (also called *reactive* [10]) systems [1, 4, 21]. A reactive system interacts with its environment, and a correct system should satisfy the specification with respect to all environments. The right way to approach synthesis of reactive systems is to consider the situation as a (possibly infinite) game between the environment and the system. More formally, a strategy for a system with inputs in I and outputs in O maps finite sequences of inputs — words in $(2^I)^*$, which correspond to the actions of the environment so far, to an output in 2^O — a suggested action for the system. A specification ψ over $I \cup O$ is then *realizable* iff there is a strategy all of whose computations satisfy ψ, whre the computation of a strategy $f : (2^I)^* \rightarrow 2^O$ on a infinite sequence $i_0, i_1, i_2, \ldots \in (2^I)^\omega$ is $i_0 \cup f(\epsilon), i_1 \cup f(i_0), i_2 \cup f(i_0 \cdot i_1), \ldots$. The *synthesis problem* for ψ is to return a finite-state transducer that realizes it (or an answer that ψ is not realizable).

While model-checking theory has led to industrial development and use of formal-verification tools, the integration of synthesis in the industry is slow. This has to do with *theoretical limitations*, like the complexity of the problem (the synthesis problem for linear temporal logic (LTL) is 2EXPTIME-complete [21]), methodological reasons (the traditional solutions to the synthesis problem require the determinization of automata on infinite words [23] and the solution of parity games [15]), and *practical reasons*: the difficulty of writing complete specifications and environment assumptions, the lack of satisfactory compositional synthesis algorithms, and suboptimal results (current algorithms produce systems that satisfy the specification, but may be larger or less well-structured than systems constructed manually, and may satisfy the specification in a peculiar way).

In the last decade there has been a significant advances in the development of practical algorithms for synthesis. In the theoretical fronts, researchers have suggested LTL synthesis algorithms that circumvent determinization and parity games [17], algorithms for fragments of LTL that can be implemented symbolically [20], and algorithms that reduce LTL synthesis to the solution of safety games [6]. These algorithms have been implemented [7, 13, 14, 20], and they also support basic compositional synthesis [7, 16]. Synthesis tools that are based on them give encouraging recent results (c.f., synthesis of an arbiter for RAM's on-chip AMBA advanced high-performance bus from temporal specifications [9], an electronic voting machine [5], and more). Work has also been done on generating environment assumptions to reduce the specification burden for synthesis [18].

In this paper we describe a new approach for solving LTL synthesis. Consider an LTL formula ψ. Like earlier approaches, our main goal is to circumvent the determinization of the automaton for ψ and the solution of parity games. Unlike earlier approaches, our algorithm is based on reducing the synthesis problem to a solution of a reachability game, played between the system and the environment on a graph obtained by combining the subset constructions of the automata for ψ and $\neg\psi$. Our algorithm is a heuristic – the goals of the system and the environment in the reachability game are not dual, and it may be that no player can force the opponent to its target states. Even in that case, the information obtained from the game enables us to restrict standard synthesis algorithms to a subset of the game, which is often much smaller. In addition, as we elaborate below, our algorithm involves theoretical issues at the heart of the synthesis problem that we believe should get

more attention. In particular, we study *synthesis with clairvoyance (look-ahead)*, which is strongly related to the need to work with deterministic automata [11, 12].

Let us now explain the idea behind our algorithm. Recall that satisfiability of an LTL formula ψ only guarantees that there is a collaborative input sequence $x \in (2^I)^\omega$ with which the system can interact and generate an output sequence $y \in (2^O)^\omega$ such that the composition of x and y into a computation in $(2^{I \cup O})^\omega$ satisfies ψ. On the other hand, in realizability, the system should have a strategy that satisfies the specification with respect to all possible environments. Between the satisfiability and the realizability problems, one can consider *universal satisfiability*, where for every input sequence $x \in (2^I)^\omega$, there is an output sequence $y \in (2^O)^\omega$ such that the composition of x and y satisfies ψ. Clearly, not all satisfiable specifications are universally satisfiable. Also, it is not hard to see that while universal satisfaction is a necessary condition for realizability, it is not a sufficient condition. A good way to understand the difference between realizability and universal satisfiability is to consider *realizability with look-ahead* – a notion that generalizes both of them. In realizability with look-ahead k, for $k \geq 0$, we also seek a strategy for the system. Here, however, the system generates the output at position j only after seeing the input in all positions up to $j + k$. It is easy to see that realizability coincides with realizability with look-ahead 0, whereas universal satisfiability coincides with realizability with look-ahead ∞.

Look-ahead helps the system in two ways. First, when the ability to satisfy the specification depends on information from the future, the look-ahead reveals the future. Second, when different futures with the same prefix require different outputs, look-ahead postpones the need to commit to the same output for both futures. One may wonder if these two ways are not two different interpretation of the same extra burden that realizability poses on universal satisfiability, and indeed this is the case. In fact, this is exactly the same burden that requires us to determinize the specification automaton in the process of solving the realizability problem: different input sequences that share the same prefix may need to follow different runs of the nondeterministic automaton, and the run may differ already in the joint prefix. A look-ahead enables us to follow different runs in the joint prefix, as long as the difference between the sequences is "in the range of visibility" of the strategy. [4]

With all this in mind, our algorithm works as follows. First, we try our luck and check whether ψ is universally satisfiable. If it is not, then clearly ψ is also non-realizable and we are done. If it is, then we again try our luck and check whether $\neg\psi$ is strongly satisfiable by the environment. If it is not, then again we are done, as we can conclude that $\neg\psi$ is not realizable by the environment, making ψ realizable by the system, and in fact it is easy to find a transducer for it – the transducer can ignore the input and just generates the output that witnesses the fact $\neg\psi$ is not universally satisfiable by the environment. Note that checking universal satisfaction is much simpler than checking realizability, not just from a theoretical point of view (the problem is EXPSPACE-complete [24]), but also in practice – universal satisfaction amounts to checking universality of a nondeterministic Büchi word automaton. Our experiments show that we may actually be lucky quite often.

Our algorithm becomes more interesting when both ψ and $\neg\psi$ are universally satisfiable. Then, we know that with an infinite look-ahead, both the system and the environment

[4] This is similar to the link between online/offline algorithms and deterministic/nondeterministic automata [2].

can satisfy their dual goals, and it is only the nature of the interaction, which requires both of them to proceed on-line, that makes only one of ψ and $\neg\psi$ realizable.[5] Consider a prefix $w \in (2^{I \cup O})^*$ of a computation. We can say that the system is *hopeful after* w if ψ stays universally satisfiable even when the interaction is restricted to start with w. Note that in the definition of universal satisfaction, the outputs are existentially quantified. Thus, fixing the outputs in w may indeed prevent ψ from being universally satisfiable. Dually, the environment is hopeful after w if $\neg\psi$ stays universally satisfiable. Our algorithm checks whether the system has a strategy to force the environment to a prefix of a computation after which only the system is hopeful, and dually for the environment. In the first case, we can conclude that ψ is realizable, and we also get a transducer for it. In the second, we know that $\neg\psi$ is realizable by the environment. The good news is that the classification of prefixes can be reduced to a sequence of checks for universal satisfaction, and is needed only for prefixes the lead to different states in the subset construction of the automata for ψ and $\neg\psi$, with no determinization needed. Also, as noted above, in case neither the system nor the environment have a strategy to make the opponent hopeless, we can restrict traditional synthesis algorithms to take into an account the need of the system and the environment to stay in a hopeful set of states. As our examples show, our algorithm often terminates with a definite answer, and it may also leads to a significant reduction in the state space. In Section 6, we also point to other advantages of our algorithm.

Finally, we study synthesis with look-ahead and describe an algorithm for solving it. A solution for the problem is described already in [12] in the context of sequential calculus, Here, we adjust the solution to the modern setting of LTL and parity games, and relate it to our heuristic. Beyond the theoretical interest in realizability with look-ahead as a notion between universal satisfiabaility and realizability, look-ahead is interesting also from a practical point of view. As we demonstrate in the paper (see also [3, 11]), look-ahead can make the difference between a specification being realizable and not being realizable. Since in practice we often do have a look-ahead (say, when the environment buffers its actions), it makes sense to use it.

2 Preliminaries

2.1 Satisfiability, universal satisfiability, and realizability

Let I and O be finite sets of input and output signals, respectively. For an input sequence $x = i_0, i_1, \ldots \in (2^I)^\omega$ and an output sequence $y = o_0, o_1, \ldots \in (2^O)^\omega$, the *computation* $x \oplus y$ is the interleaved sequence $i_0 \cup o_0, i_1 \cup o_1, \ldots \in (2^{I \cup O})^\omega$.

Consider an LTL formula ψ over $I \cup O$. We consider three levels of satisfaction of ψ.

- The formula ψ is *satisfiable* if there is a computation that satisfies ψ.
- The formula ψ is *universally satisfiable* if for every input sequence $x \in (2^I)^\omega$, there is an output sequence $y \in (2^O)^\omega$ such that $x \oplus y$ satisfies ψ.
- The formula ψ is *realizable* if there is a strategy $f : (2^I)^* \to 2^O$ such that for every input sequence $x = i_0, i_1, i_2, \ldots \in (2^I)^\omega$, the computation of f on x, that is $i_0 \cup f(\epsilon), i_1 \cup f(i_0), i_2 \cup f(i_0 \cdot i_1), \ldots$ satisfies ψ.

[5] An orthogonal research direction is to study the cases in which this happens, and the setting in which a bounded lookahead is sufficient. As shown in [11], such problems are decidable.

It is not hard to see that realizability implies universal satisfiability, which implies satisfiability, but not the other way around. For example, let $I = \{q\}$ and $O = \{p\}$. It is easy to see that the formula Gq is satisfiable but not universally satisfiable. Also, the formula $G(p \leftrightarrow q)$ is universally satisfiable but not realizable. Indeed, if, by way of contradiction, f is a strategy that realizes it, then an input sequence x that starts with q if $f(\epsilon) = \emptyset$ and starts with $\{\emptyset\}$ if $f(\epsilon) = \{p\}$ is such that the computation of f on x does not satisfy $p \leftrightarrow q$, and hence does not satisfy $G(p \leftrightarrow q)$.

We note that in our definition of realizability, we did not require the strategy f to be finite state. Since LTL formulas induce regular languages, adding such a requirement would result in an equivalent definition [22]. Formally, a strategy $f : (2^I)^* \to 2^O$ is finite state if for every $o \in 2^O$, the language $f^{-1}(o)$, which is a subset of $(2^I)^*$, is regular. Equivalently, f is finite state if it is induced by a finite-state transducer – a deterministic automaton over the alphabet 2^I in which each state is labeled by a letter in 2^O. Then, given a sequence $w \in (2^I)^*$, the strategy f induced by the transducer is such that $f(w)$ is the label of the state that the transducer visits after reading w.

2.2 Automata on infinite words

A specification over $I \cup O$ can be viewed as a language over the alphabet $2^{I \cup O}$. The decision procedures for the three levels of satisfaction discussed above follow this view, and are based on automata on infinite words.

A *nondeterministic automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha \rangle$, where Σ is a finite nonempty alphabet, Q is a finite nonempty set of states, $Q_0 \subseteq Q$ is a nonempty set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, and α is an acceptance condition. The automaton $\mathcal{A}$ is *deterministic* if $|Q_0| = 1$ and $|\delta(q, \sigma)| \leq 1$ for all states $q \in Q$ and symbols $\sigma \in \Sigma$.

A *run* r of $\mathcal{A}$ on an infinite word $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$ is an infinite sequence $q_0, q_1, \ldots$ of states such that $q_0 \in Q_0$, and for all $i \geq 0$, we have $q_{i+1} \in \delta(q_i, \sigma_{i+1})$. The acceptance condition α determines which runs are accepting. In the *Büchi* acceptance condition, $\alpha \subseteq Q$, and a run r is accepting if it visits some state in α infinitely often. Formally, let $inf(r) = \{q : q_i = q \text{ for infinitely many } i\text{'s}\}$. Then, r is *accepting* iff $inf(r) \cap \alpha \neq \emptyset$. A word w is accepted by an automaton $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. The *language* of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. We say that $\mathcal{A}$ is *empty* if $L(\mathcal{A}) = \emptyset$ and that $\mathcal{A}$ is *universal* if $L(\mathcal{A}) = \Sigma^\omega$. A *pre-automaton* is an automaton without an acceptance condition. We use NBW and DBW to abbreviate nondeterministic and deterministic Büchi automata, respectively.

We are going to mention also the *co-Büchi* and the *parity* acceptance conditions. The condition co-Büchi is dual to Büchi, thus a run is accepting if it visits α only finitely often. The parity is more complicated and for our purposes here it is enough to note that determistic parity automata (DPWs) are sufficiently expressive to recognize all the languages recognized by nondeterministic Büchi automata. Thus, NBWs can be translated to DPWs [19, 23].

Theorem 1. [25] *For every LTL formula ψ, there is an NBW $\mathcal{A}_\psi$ with $2^{O(|\psi|)}$ states such that $L(\mathcal{A}_\psi) = \{w : w \models \psi\}$.*

2.3 Traditional decision procedures

In this section we briefly review the traditional algorithms for solving satisfiability, universal satisfiability, and realizability.

Deciding satisfiability is PSPACE-complete: given ψ, one can follow Theorem 1 and constructs the NBW $\mathcal{A}_\psi$. Clearly, ψ is satisfiable iff $L(\mathcal{A}_\psi)$ is not empty. Since the size of $\mathcal{A}_\psi$ is exponential in the length of ψ and checking its nonemptiness can be done on-the-fly in NLOGSPACE, the PSPACE complexity follows.

Deciding universal satisfiability is more complicated and is EXPSPACE-complete: Starting with $\mathcal{A}_\psi$, we construct an NBW $\mathcal{A}_\psi^{\exists O}$, obtained from $\mathcal{A}_\psi$ by taking its projection on I. That is, if $\mathcal{A} = \langle 2^{I \cup O}, Q, Q_0, \delta, \alpha \rangle$, then $\mathcal{A}_\psi^{\exists O} = \langle 2^I, Q, Q_0, \delta^{\exists O}, \alpha \rangle$, where for a state $q \in Q$ and input $i \in 2^I$, we have that $\delta^{\exists O}(q, i) = \{s : \exists o \in 2^O \text{ such that } s = \delta(q, i \cup o)\}$. It is not hard to see that a word $x \in (2^I)^\omega$ is accepted by $\mathcal{A}_\psi^{\exists O}$ iff there is a word $y \in (2^O)^\omega$ such that $x \oplus y$ is accepted by $\mathcal{A}$. Hence, $\mathcal{A}_\psi^{\exists O}$ is universal iff ψ is strongly satisfiable. Checking the universality of $\mathcal{A}_\psi^{\exists O}$ can be done by checking the emptiness of its complement. Since the size of $\mathcal{A}_\psi$, and hence also of $\mathcal{A}_\psi^{\exists O}$ is exponential in the length of ψ, complementation involves an exponential blow-up, and emptiness can be checked in NLOGSPACE, the EXPSPACE complexity follows.

Finally, deciding realizability is even more complicated, and is 2EXPTIME-complete. The traditional algorithm determinizes $\mathcal{A}_\psi$, and transforms the obtained DPW into a two-player game between the system and the environment. Formally, let $\mathcal{D}_\psi = \langle 2^{I \cup O}, Q, q_0, \delta, \alpha \rangle$ be the DPW for ψ. Then, the game is $G_\psi = \langle V, E \rangle$, where the set of vertices $V = V_{sys} \cup V_{env}$ is such that $V_{sys} = Q$ and $V_{env} \subseteq 2^Q$. For $S \in 2^Q$, we have that $S \in V_{env}$ iff there is $q \in Q$ and $o \in 2^O$ such that $S = \delta^{\exists I}(q, o)$, in which case $E(q, S)$. Also, $E(S, q')$ iff $q' \in S$. Deciding the realizability problem then amounts to deciding the winner in the game G_ψ with winning objective α. Intuitively, each transition of $\mathcal{D}_\psi$ is partitioned in the game G_ψ into two transitions: consider a vertex $q \in V_{sys}$. First, the system chooses an output $o \in 2^O$, and the game moves to the vertex $\delta^{\exists I}(q, o) \in V_{env}$. Then, the environment chooses an input $i \in 2^I$ and the game continues to the state in $\delta^{\exists I}(q, o)$ that i leads to, namely to $\delta(q, i \cup o) \in V_{sys}$.

It is sometimes convenient to refine G_ψ to include more information, which enables a labeling of the edges by the actions taken by the players. Thus, here $E \subseteq (V_{sys} \times 2^O \times V_{env}) \cup (V_{env} \times 2^I \times V_{sys})$. For that, we define, $V_{sys} = Q$ and $V_{env} \subseteq Q \times 2^O \times 2^Q$ is such that $\langle q, o, S \rangle \in V_{env}$ iff $S = \delta^{\exists I}(q, o)$. Then, we also have $E(q, o, \langle q, o, S \rangle)$. In addition, for all vertices $\langle q, o, S \rangle \in V_{env}$ and $q' \in V_{sys}$, we have that $E(\langle q, o, S \rangle, i, q')$ iff $q' = \delta(q, i \cup o)$. Note that $q' \in S$.

The system and the environment are dual, in the sense that we can view the setting as one in which the environment is trying to satisfy $\neg\psi$ when it interacts with all systems. Thus, the roles of the system and the environment may be switched, and we can talk about a formula ψ being universally satisfied by the environment, meaning that for every output sequence $y \in (2^O)^\omega$, there is an input sequence $x \in (2^I)^\omega$ such that $x \oplus y$ satisfies ψ. We can also talk about ψ being realizable by the environment, meaning that there is a finite-state strategy $g : (2^O)^* \to 2^I$ such that for every output sequence $y = o_0, o_1, o_2, \ldots \in (2^O)^\omega$, the computation of g on y, that is $o_0 \cup g(o_0), o_1 \cup g(i_0 \cdot o_1), o_2 \cup g(o_0 \cdot o_1 \cdot o_2), \ldots$ satisfies ψ. Note that in both types of realizability (by the system and by the environment), the system moves first. Thus, the settings are not completely dual.

For universal satisfiability, the identity of the player that moves first is irrelevant, and the definitions are completely dual.[6] From determinancy of games, we know that either ψ is realizable by the system or $\neg\psi$ is realizable by the environment.

3 Using universal satisfiability

In this section we describe the first steps in our methodology for using universal satisfiability in the process of checking realizability. We also point to realizability with look-ahead as a notion between universal satisfiability and realizability.

Given a property ψ over I and O, we proceed as follows.

(1) Check universal satisfiability of ψ.
- **(1.1)** If the answer is negative, we are done. Indeed, if ψ is not universally satisfiable, then clearly ψ is also not realizable.
- **(1.2)** If the answer is positive, proceed to (2).

(2) Check universal satisfiability of $\neg\psi$ by the environment.
- **(2.1)** If the answer is negative, we are done. Indeed, if $\neg\psi$ is not universally satisfiable by the environment, then clearly $\neg\psi$ is also not realizable by the environment, implying that ψ is realizable by the system. Moreover, a transducer for ψ can simply generate the output sequence $y \in (2^O)^\omega$ for which for all $x \in (2^I)^\omega$ we have that $x \oplus y \models \psi$.
- **(2.2)** If the answer is positive, proceed to (3).

(3) This is the interesting case: both ψ and $\neg\psi$ are universally satisfiable. Note that while it cannot be that both ψ and $\neg\psi$ are realizable, they can both be universally satisfiable. When this happens, we know that one of the players, the system or the environment, cannot arrange the responses that work for the universal satisfiability in the form of the strategy that is needed for realizablity. For example, consider the formula $\psi = G(p \leftrightarrow q)$, with $I = \{q\}$ and $O = \{p\}$. Note that $\neg\psi = F(\neg(p \leftrightarrow q))$. While both ψ and $\neg\psi$ are universally satisfiable, only $\neg\psi$ is realizable by the environment.

The example of a robotic vehicle controller from [18], demonstrates how our heuristic detects that the system is not realizable when sufficient assumptions are not provided. The example of the robotic vehicle controller aims to synthesize a discrete planner that allows an autonomous robot to move in a rectangular grid, while avoiding obstacles. The obstacles are put and cleared by the environment at arbitrary times and squares. In this example, the specification ψ is of the form $\mathbf{A} \rightarrow \mathbf{G}$, where $\mathbf{A}$ is a conjunction of assumptions on the environment, and $\mathbf{G}$ is a conjunction of guarnatees. The guarantees require the car to start at the initial square, and in each step to move to an adjacent square or to stay in the current one. The car cannot move to an occupied square, and it eventually have to reach the destination square. The assumptions on the environment require that there are no obstacles at the initial and destination squares. With this weak assumption, ψ is not universally satisfied and our heuristic terminates at Step **(1.1)**. In order to make the specification realizble, we need to add stronger assumptions to $\mathbf{A}$. Adding the assumption that " all the squares must be clear of obstacles infinitely often" resolves the problem, and

[6] The cleanest way to handle this lack of duality is to parameterize the synthesis problem with a "who moves first" flag. We decided to keep the setting simpler and let the system move first in both settings.

makes ψ realizable. Here too, out heuristic is helpful, as with the stronger assumption we get that $\neg\psi$ is not universally satisfied by the environment, thus our heuristic terminates at Step **(1.2)**.

Before we proceed to describe how our algorithm continues in Step (3), let us discuss the situation in more detail. Consider again the formula $\psi = G(p \leftrightarrow q)$. As noted above, ψ is not realizable. Intuitively, once the system generates an output, the environment can generate an input that does not agree with the polarity of the output, thus violating the specification. But what if the system can generate its output only after seeing the next input? Then, the specification is realizable. In general, the difference between universal satisfiability and realizability is the fact that in universal satisfiability the system knows the whole sequence of inputs before generating the output, whereas in realizability, the system has to react online and generate the next output without knowing the inputs yet to come. Between these two extreme cases, we can talk about *realizability with look-ahead*, where the system has to generate the next output after seeing a prefix of the inputs yet to arrive.

Definition 1. [realizable with look-ahead] *An LTL formula ψ over $I \cup O$ is realizable with look-ahead k (k-realizable, for short), if there is a strategy $f : (2^I)^{\geq k} \to 2^O$ such that for every input sequence $w = i_0, i_1, i_2, \ldots \in (2^I)^\omega$, the computation of f on w, that is $i_0 \cup f(i_0), i_1 \cup f(i_0, i_1), i_2 \cup f(i_0, i_1, i_2), \ldots, i_j \cup f(i_0, i_1, \ldots, i_{k+j-1}), \ldots$ satisfies ψ.*

As explained in Section 1, both universal satisfiability and realizability are a special cases of k-realizability; the first with $k = \infty$ and the second with $k = 0$. Also, realizability with look-ahead is interesting also in practice, as it corresponds to realistic settings and can make specifications realizable [11, 12].

4 When both ψ and $\neg\psi$ are universally satisfiable

In this section we continue the description of our algorithm, namely what to do when we get to Step (3). Let $\mathcal{A}_\psi = \langle 2^{I\cup O}, S, S_0, \rho, \alpha\rangle$ and $\mathcal{A}_{\neg\psi} = \langle 2^{I\cup O}, S', S'_0, \rho', \alpha'\rangle$ be NBWs for ψ and $\neg\psi$, respectively. Let $\mathcal{U}_\psi$ be the pre-automaton obtained by applying the subset construction to $\mathcal{A}_\psi$ and $\mathcal{A}_{\neg\psi}$. Thus, $\mathcal{U}_\psi = \langle 2^{I\cup O}, 2^S \times 2^{S'}, \langle S_0, S'_0\rangle, \delta\rangle$, where for all $\langle P, P'\rangle \in 2^S \times 2^{S'}$ and $\sigma \in 2^{I\cup O}$, we have that $\delta(\langle P, P'\rangle, \sigma) = \langle \rho(P, \sigma), \rho'(P', \sigma)\rangle$. For a state $\langle P, P'\rangle$ of $\mathcal{U}_\psi$, let $L(\mathcal{A}^P_\psi)$ and $L(\mathcal{A}^P_{\neg\psi})$ be the languages of $\mathcal{A}_\psi$ and $\mathcal{A}_{\neg\psi}$ with initial sets P and P', respectively. We say that a set $P \in 2^S$ is *system hopeful* (sys-hopeful, for short) if for all $x \in (2^I)^\omega$ there is $y \in (2^O)^\omega$ such that $x \oplus y \in L(\mathcal{A}^P_\psi)$. We say that a set $P' \in 2^{S'}$ is *environment hopeful* (env-hopeful, for short) if for all $y \in (2^O)^\omega$ there is $x \in (2^I)^\omega$ such that $x \oplus y \in L(\mathcal{A}^{P'}_{\neg\psi})$. Thus, system hopefulness coincides with universal satisfaction, except that instead of talking about satisfaction of an LTL formula we talk about the membership in the language of $\mathcal{A}^P_\psi$. Dually, environment hopefulness refer to membership in $\mathcal{A}^{P'}_{\neg\psi}$.

Consider a state $\langle P, P'\rangle \in 2^S \times 2^{S'}$ of $\mathcal{U}_\psi$. It is possible to decide in space exponential in the length of ψ whether P is system hopeful and whether P' is environment hopeful. Indeed, the check is similar to the check for universal satisfaction described in Section 2. For the case of system hopefulness, we project $\mathcal{A}^P_\psi$ on 2^I and check that the

obtained NBW is universal. For environment hopefulness we do the same, with $\mathcal{A}^{P'}_{\neg\psi}$ and a projection on 2^O.

Remark 1. In case we start with a deterministic automaton $\mathcal{D}_\psi$ for the specification, we do not have to apply the subset construction, and we can work directly with $\mathcal{D}_\psi$. Then, the notion of system and environment hopefulness applies to single states, and checking whether a state s is env-hopeful is done by dualizing $\mathcal{D}_\psi$, thus getting a deterministic co-Büchi automaton for the negation of ψ. We can then project the co-Büchi automaton existentially on 2^O, and check whether the result is universal (see Example 1).

We can now describe the continuation of the algorithm:

(3) Consider the game induced by the pre-automaton $\mathcal{U}_\psi$.

(3.1) If the system has a strategy to reach a state $\langle P, P'\rangle$ such that P is sys-hopeful and P' is not env-hopeful, then we are done. Indeed, ψ is realizable, and we can also have a transducer for it.

(3.2) If the environment has a strategy to reach a state $\langle P, P'\rangle$ such that P' is env-hopeful and P is not sys-hopeful, then we are done. Indeed, in a manner dual to the one above, $\neg\psi$ is realizable by the environment.

(3.3) If we got here, both the system and the environment have strategies to stay forever in the region of states that are both sys-hopeful and env-hopeful. At this point we give up and turn to solve the realizability problem using one of the traditional algorithms. The information gathered during our algorithm is still useful and enables us to restrict the realizability game to states in the region of hopeful states (all the other states are replaced by two states – one is winning for the system and one is winning for the environment).

We conclude the description of the algorithm with the following theorem.

Theorem 2. *Consider an LTL specification ψ over $I \cup O$.*

1. *If the algorithm reaches Step* **(3)***, then all the states $\langle P, P'\rangle$ that are reachable in $\mathcal{U}_\psi$ are such that at least one of the sets P and P' is hopeful.*
2. *If the algorithm terminates in Steps* **(2.1)** *or* **(3.1)***, then ψ is realizable and the checks done by the algorithm induce a transducer for the system that satisfies ψ.*
3. *If the algorithm terminates in Steps* **(1.1)** *or* **(3.2)***, then ψ is not realizable and the check done induce a transducer for the environment that satisfies $\neg\psi$.*

Proof: We start with the first point. Consider a state $\langle P, P'\rangle$ that is reachable in $\mathcal{U}_\psi$. Let w be a word that leads to $\langle P, P'\rangle$. Consider now the parity game that corresponds to the realizability problem for ψ and the vertex v_w that the game reaches after the system and the environment proceeds according to w. Since parity games are determined, v_w is a winning vertex for either the system (in which case P must be hopeful) or the environment (in which case P' must be hopeful).

Now, if the algorithm terminates Step (2.1), then ψ is realizable as a transducer for it can simply generate the output sequence $y \in (2^O)^\omega$ for which for all $x \in (2^I)^\omega$ we have that $x \oplus y \models \psi$; the fact $\neg\psi$ is not universally realizable by the environment guarantees that such a sequence y exists, and we know how to find it: this is the sequence that witnesses the nonemptiness of the complement of $\mathcal{A}^{\exists I}_{\neg\psi}$.

Finally, when the algorithm terminates in Step (3.2), then ψ is realizable as a transducer for it can start with the strategy that reaches $\langle P, P' \rangle$ such that P is sys-hopeful and P' is not env-hopeful. It is guaranteed that when we apply Steps (1+2) of the algorithm with $\mathcal{A}^P_\psi$ instead of ψ and $\mathcal{A}^{P'}_{\neg\psi}$ instead of $\neg\psi$, we would end up end up in Step (2.1), thus once generating the prefix that leads to $\langle P, P' \rangle$, the transducer can continue with a fixed output sequence, as described above. The case ψ is not realizable is dual. □

We now demonstrate our algorithm with three examples. In all of them, we have for $I = \{q\}$ and $O = \{p\}$.

Example 1. We start with an example in which the NBW for the specification is deterministic. Let $\psi = G(p \leftrightarrow Fq)$. Note that ψ is equivalent to $G(p \rightarrow Fq) \wedge G(\neg p \rightarrow G\neg q)$. The specification is universally satisfiable: given a sequence $x \in (q, \neg q)^\omega$, it is not hard to see that the sequence $y \in (p, \neg p)^\omega$ in which p holds in position j iff q holds in a position grater than j is such that $x \oplus y \models \psi$. Consider the negation of the specification, that is $\neg\psi = F(p \wedge G\neg q) \vee F(\neg p \wedge Fq)$. It is not hard to see that $\neg\psi$ is universally satisfiable by the environment. Indeed, given a sequence $y \in (p, \neg p)^\omega$, a sequence $x \in (q, \neg q)^\omega$ in which q holds exactly when p does not hold, is such that $x \oplus y \models \neg\psi$.

On the left of Figure 1 below we describe a DBW $\mathcal{D}_\psi$ for ψ. Since $\mathcal{D}_\psi$ is deterministic, we do not have to apply the subset construction on it. On the right, we describe the two projections of $\mathcal{D}_\psi$ on I and on O.

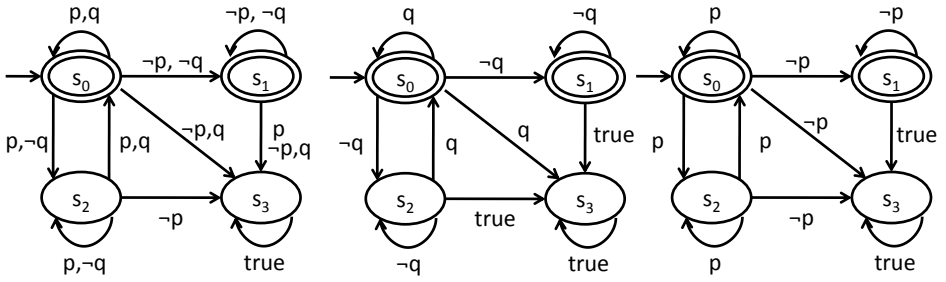

Fig. 1. A DBW $\mathcal{D}_\psi$ for $\psi = G(p \leftrightarrow Fq)$ (left), and its projections $\mathcal{D}^{\exists O}_\psi$ and $\mathcal{D}^{\exists I}_{\neg\psi}$ on I (middle) and O (right), respectively.

Note that in $\mathcal{D}^{\exists O}_\psi$, only s_0 is universal (the other states are not universal since, for example, q^ω is not accepted from them). Thus, only s_0 is sys-hopeful in $\mathcal{D}_\psi$. In order to find the env-hopeful states we consider the co-Büchi automaton $\mathcal{D}^{\exists I}_{\neg\psi}$. Here, all states are universal. Indeed, s_3 is an accepting sink, s_1 can get with both p and $\neg p$ to s_3 in one transition, s_2 can stay in s_2 forever with p^ω, and with all other words it can reach s_3, and finally, s_0 can reach s_2 and s_1 with p and $\neg p$, respectively. It follows that all the states in $\mathcal{D}_\psi$ are env-hopeful.

The game induced by $\mathcal{D}_\psi$ appears in Figure 2. The system states are ovals (and in them, the system chooses between p and $\neg p$), and the environment states are rectangles (the environment chooses q or $\neg q$). It is not hard to see that the environment has a strategy

to force the system to a state that is not sys-hopeful while staying within env-hopeful states. Thus, we can conclude that ψ is not realizable.

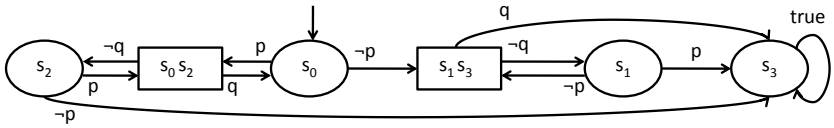

Fig. 2. The game induced by $\mathcal{D}_\psi$.

Example 2. We now consider a case where $\mathcal{A}_\psi$ is nondeterministic, thus we proceed with NBWs for both ψ and $\neg\psi$. Consider the specification $\psi = (Gp \wedge Fq) \vee (G\neg p \wedge F\neg q)$. Thus, either the system always generates p and the environment generates q eventually, or the system always generates $\neg p$, and the environment generates $\neg q$ eventually. Note that $\neg\psi = (Fp \wedge F\neg p) \vee (G\neg q \wedge Fp) \vee (Gq \wedge F\neg p)$. It is not hard to see that ψ is universally satisfiable by the system and $\neg\psi$ is universally satisfiable by the environment.

In Figure 3, we describe the NBWs $\mathcal{A}_\psi$ (on the left, a union of two components) and $\mathcal{A}_{\neg\psi}$ (on the right, a union of three components).

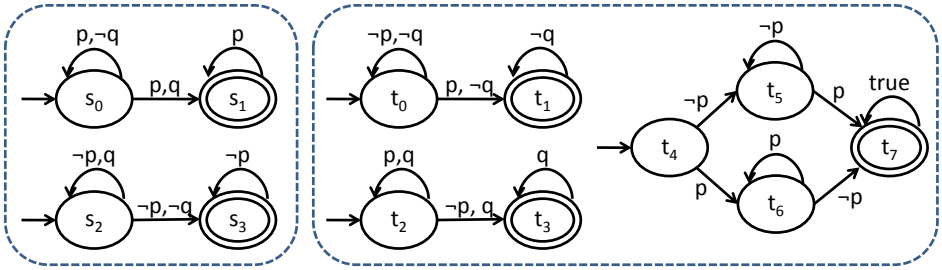

Fig. 3. The NBWs $\mathcal{A}_\psi$ (left) and $\mathcal{A}_{\neg\psi}$ (right).

We now check the system and environment hopefulness of sets that are reachable in the subset construction of the two NBWs. If we get to a set that is not hopeful, there is no need to continue the construction from it. In Figure 4 we describe the obtained deterministic pre-automata. In the figure, we indicate by dashed lines that the set is not hopeful. For example, the set $\{s_0\}$ is not sys-hopeful since there is no output sequence $y \in (p, \neg p)^\omega$ such that $x \oplus y$ is accepted from $\mathcal{A}_\psi^{\{s_0\}}$ for $x = (\neg q)^\omega$. Similarly, the set $\{t_0, t_5\}$ is not env-hopeful since there is no input sequence $x \in (q, \neg q)^\omega$ such that $x \oplus y$ is accepted from $\mathcal{A}_{\neg\psi}^{\{t_0,t_5\}}$ for $y = (\neg q)^\omega$.

In Figure 5 we describe the game corresponding to $\mathcal{U}_\psi$, obtained by combining the two pre-automata. As indicated in the figure, the states $(\{s_1\}, \{t_2, t_6\})$ and $(\{s_3\}, \{t_0, t_5\})$

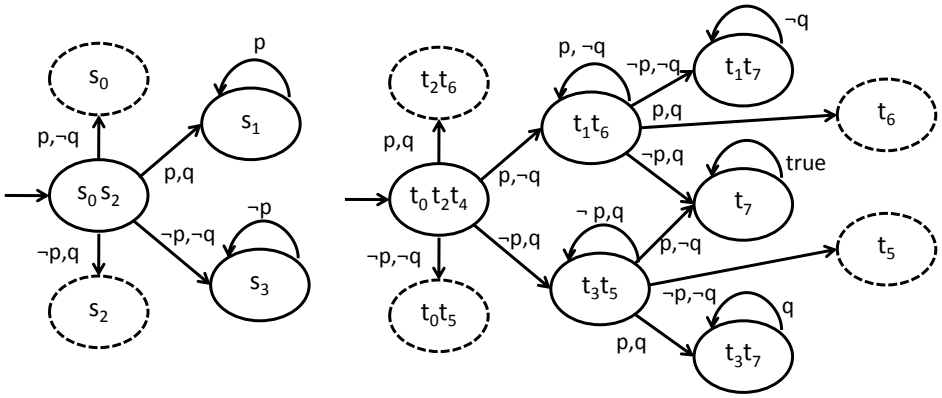

Fig. 4. The pre-automata obtained by applying the subset construction to $\mathcal{A}_\psi$ (left) and $\mathcal{A}_{\neg\psi}$ (right).

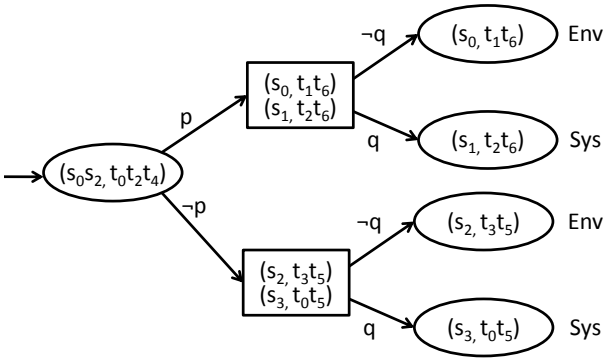

Fig. 5. The game corresponding to $\mathcal{U}_\psi$.

are winning states for the system. Indeed, $\{s_1\}$ is sys-hopeful whereas $\{t_2, t_6\}$ is not env-hopeful, and likewise, $\{s_3\}$ is sys-hopeful whereas $\{t_0, t_5\}$ is not env-hopeful. Dually, the states $(\{s_0\}, \{t_1, t_6\})$ and $(\{s_2\}, \{t_3, t_5\})$ are winning states for the environment. It is not hard to see that the environment has a strategy to reach its winning states, thus we conclude that ψ is not realizable.

Example 3. In this example we demonstrate a case in which our algorithm does not reach a definite answer. Consider the specification $\psi = F(p \leftrightarrow q)$. Again, both ψ and $\neg\psi$ are universally satisfiable, so we get to Step (3). The deterministic automaton $\mathcal{D}_\psi$ of ψ appears in Figure 6. It is easy to see that both s_0 and s_1 are system hopeful, whereas only s_0 is environment hopeful. However, the system does not have a strategy to force the game induced by $\mathcal{D}_\psi$ to s_1: if the system proceeds from s_0 with p, the environment will respond with $\neg q$, and if the system proceeds with $\neg p$, the environment will respond with q. Also, since both s_0 and s_1 are system hopeful, the environment does not have a strategy to force the game into states that are not system hopeful. So, we have to solve

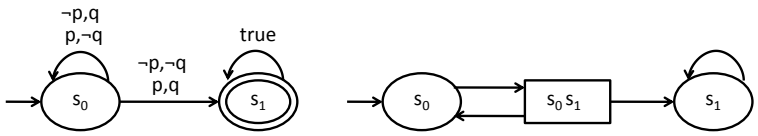

Fig. 6. A DBW for $\psi = F(p \leftrightarrow q)$ and the game corresponding to it.

the realizability problem. We can use, however, the fact that game would get stuck in s_0, which would not satisfy the Büchi condition, thus ψ is not realizable.

5 LTL realizability with look-ahead

In Section 3, we defined LTL realizability with look-ahead. This notion is not fundamentally new, and the problem of k-realizability was studied already in 1972, in the context of sequential calculus [12]. Here, we adjust the solution to the modern setting of LTL and DPWs, and describe how our algorithm can be adjusted to handle k-realizability too.

Theorem 3. *Consider an LTL formula ψ and an integer $k \geq 0$. Let $\mathcal{A}^k_\psi$ be such that $L(\mathcal{A}^k_\psi) = \{x \oplus y : x \oplus y^k \in L(\mathcal{A}_\psi)\}$, where y^k is the suffix of y from position k.*

- *We can construct an NBW $\mathcal{A}^k_\psi$ as above with number of states exponential in $|\psi|$ and k.*
- *We can construct an DPW $\mathcal{A}^k_\psi$ as above with number of states doubly-exponential in $|\psi|$ and exponential in k.*
- *Applying synthesis algorithms with respect to $\mathcal{A}^k_\psi$ rather than $\mathcal{A}_\psi$ solves the k-realizability problem.*

Proof: Let $\mathcal{A}_\psi = \langle 2^{I \cup O}, Q, \delta, q_0, \alpha \rangle$ be an NBW for ψ. We define $\mathcal{A}^k_\psi = \langle 2^{I \cup O}, Q', \epsilon, \delta', \alpha' \rangle$, where

- $Q' = (\bigcup_{0 \leq j < k} (2^I)^j) \cup (Q \times (2^I)^k)$. The first type of states is for accumulating the vector of the last $k-1$ inputs. The second type is to be used after we have accumulated the first k inputs. Then, we follow the runs of $\mathcal{A}_\psi$, with the output being combined with the input read k letters earlier.
- The transition function is defined as follows.
  - For the first type of states, if $0 \leq j < k-1$, we ignore the output component of the letter read (intuitively, since we shift the output by k, the output in the first $k-1$ levels is not important) and only accumulate inputs in the vector. Accordingly, $\delta'(\langle i_1, \ldots, i_j \rangle, i \cup o) = \{\langle i_1, ..., i_j, i \rangle\}$.
  - In the last level of states from the first type, we still ignore the output read but get ready to start following the runs of $\mathcal{A}_\psi$. Accordingly, $\delta'(\langle i_1, \ldots, i_{k-1} \rangle, i \cup o) = Q_0 \times \{\langle i_1, ..., i_{k-1}, i \rangle\}$.
  - Then, we continue to follow the runs of $\mathcal{A}_\psi$, where o is combined with the input read k transitions earlier. Accordingly, $\delta'(\langle q, i_1, \ldots, i_k \rangle, i \cup o) = \delta(q, i_1 \cup o) \times \{\langle i_2, ..., i_k, i \rangle\}$.
- α' is obtained from α by replacing a set $F \subseteq Q$ by the set $F \times (2^I)^k$.

The construction of the DPW $\mathcal{A}^k_\psi$ is similar, starting from a DPW $\mathcal{A}_\psi$ for the property. Note that we could have also determinized the NBW described above, but the blow-up in terms of I could then have been doubly exponential. Note that $\mathcal{A}^k_\psi$ proceeds according to the input that was read k positions earlier, combined with the current output. This captures the fact that in k-realizability, the output is combined with the input only after knowing what the previous k inputs were. Accordingly, the game induced by the DPW $\mathcal{A}^k_\psi$ solves the k-realizability problem. □

Applying our algorithm in order to solve the k-realizability problem, we proceed with the game obtained from the subset construction applied on the the NBWs $\mathcal{A}^k_\psi$ and $\mathcal{A}^k_{\neg\psi}$. Note that in both automata, the O-component of a letters is combined with the I-component of the letter read k positions earlier. All the other details of the algorithm are the same.

6 Discussion

We described a simple heuristic that replaces the parity game corresponding to LTL synthesis with a game in which the system and the environment try to force each other into hopeless states in the game. Our definition of hopeless is based on universal satisfaction – the game-free variant of realizability, and is therefore easier to reason about.

Below we discuss some further advantages of our heuristic, and some directions for future research. First, several challenges in the context of realizability are easier to cope with using our approach. This includes compositional synthesis [16], mining for assumptions [18], and testing for inherent vacuity in specifications [8]. In all these problems, one can try to circumvent the need to work with parity games by using our heuristic that use instead hopeless finite prefixes.

Our definition of hopeful states can be replaced by other definitions, leading to looser (but even more efficient) or tighter (but more complex to achieve) heuristics. On the loose side, one can work with the nondeterministic automaton (rather than the subset construction on it). Under this definition, a prefix of a computation is hopeful if there is a single state s in $\mathcal{A}_\psi$ such that the prefix can lead to s and $\mathcal{A}^s_\psi$ is universally satisfiable. Note that now, states that are not hopeful may still be reachable by hopeful prefixes, thus the heuristic can be used in order to direct the subset construction to construct subsets only when such a construction is needed. On the tighter side, one can replace universal satisfaction by definitions that are game-based, but are easier to solve than parity.

Finally, in case our algorithm does not terminate with a definite answer, we suggested to continue with traditional synthesis algorithms, with actions being restricted to these that keep the system and the environment in their hopeful regions. We found this case very interesting: both the system and the environment can stay hopeful forever, yet only one of them can satisfy the acceptance condition of $\mathcal{A}_\psi$ (the system) or $\mathcal{A}_{\neg\psi}$ (the environment). We plan to study whether this special situation can be of help when we solve the parity game on the restricted region.

References

1. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 25th ICALP*, LNCS 372, pages 1–17, 1989.

2. B. Aminof, O. Kupferman, and R. Lampert. Reasoning about Online Algorithms with Weighted Automata. In *Proc. 20th SODA*, pages 835–844, 2009.
3. D. Breslauer. On competitive on-line paging with lookahead. *TCS*, 209(1–2):365–375, 1998.
4. D.L. Dill. *Trace theory for automatic hierarchical verification of speed independent circuits*. MIT Press, 1989.
5. L. Dworkin, W. Li, and S.A. Seshia. Automatic synthesis of a voting machine design. Unpublished manuscript, 2010.
6. E. Filiot, N. Jin, and J.-F. Raskin. An antichain algorithm for LTL realizability. In *Proc. 21st CAV*, LNCS 5643, pages 263–277, 2009.
7. E. Filiot, N. Jin, and J.-F. Raskin. Compositional algorithms for LTL synthesis. In *Proc. 8th ATVA*, LNCS 6252, pages 112–127, 2010.
8. D. Fisman, O. Kupferman, S. Sheinvald, and M.Y. Vardi, A Framework for Inherent Vacuity. In *Proc. HVC08*, LNCS 5394, pages 7–22, 2008.
9. Y. Godhal, K. Chatterjee, and T.A. Henzinger. Synthesis of AMBA AHB from formal specification. CoRR abs/1001.2811, 2010.
10. D. Harel and A. Pnueli. On the development of reactive systems. In volume F-13 of *NATO Advanced Science Institutes*, pages 477–498. Springer, 1985.
11. M. Holtmann, L. Kaiser, and W. Thomas. Degrees of lookahead in regular infinite games. In *Proc. 13th FoSSaCS*, LNCS 6014, pages 252–266. Springer, 2010.
12. F. Hosch and L. Landweber. Finite delay solutions for sequential conditions. In *Proc. 1st ICALP*, page 4560, 1972.
13. B. Jobstmann and R. Bloem. Game-based and simulation-based improvements for LTL synthesis. In *Proc. 3nd GDV*, 2006.
14. B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem. Anzu: A tool for property synthesis. In *Proc. 19th CAV*, LNCS 4590, pages 258–262, 2007.
15. M. Jurdzinski, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM Journal on Computing*, 38(4):1519–1532, 2008.
16. O. Kupferman, N. Piterman, and M.Y. Vardi. Safraless compositional synthesis. In *Proc. 18th CAV*, LNCS 414, pages 31–44, 2006.
17. O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th FOCS*, pages 531–540, 2005.
18. W. Li, L. Dworkin, and S.A. Seshia. Mining assumptions for synthesis. In *Proc. 9th MEMOCODE*, July 2011.
19. N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st LICS*, pages 255–264, 2006.
20. N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive(1) designs. In *Proc. 7th VMCAI*, LNCS 3855, pages 364–380, 2006.
21. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.
22. M.O. Rabin. Automata on infinite objects and Church's problem. *Amer. Mathematical Society*, 1972.
23. S. Safra. On the complexity of ω-automata. In *Proc. 29th FOCS*, pages 319–327, 1988.
24. A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.
25. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.