

# Recent Challenges and Ideas in Temporal Synthesis

Orna Kupferman

Hebrew University, School of Engineering and Computer Science, Jerusalem 91904, Israel  
Email: orna@cs.huji.ac.il

**Abstract.** In automated synthesis, we transform a specification into a system that is guaranteed to satisfy the specification against all environments. 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, algorithmic limitations, like the need to determinize automata on infinite words and solve parity games, methodological reasons, like the lack of satisfactory compositional synthesis algorithms, and practical reasons: current algorithms produce systems that satisfy the specification, but may do so in a peculiar way and may be larger or less well-structured than systems constructed manually.

The research community has managed to suggest some solutions to these limitations, and bring synthesis algorithms closer to practice. Significant barriers, however, remain. Moreover, the integration of synthesis in real applications has taught us that the traditional setting of synthesis is too simplified and has brought with it new algorithmic challenges. This paper introduces the synthesis problem, algorithms for solving it, and recent promising ideas in making temporal-synthesis useful in practice.

## 1 Introduction

One of the most significant developments in the area of system verification over the last two decades has been the development of algorithmic methods for verifying temporal specifications of finite-state systems; see [13]. A frequent criticism against this approach, however, is that verification is done *after* significant resources have already been invested in the development of the system. Since systems invariably contain errors, verification simply becomes part of the debugging process. The critics argue that the desired goal is to use the specification of the system in the development process in order to guarantee the design of correct systems. This is called *system synthesis*.<sup>1</sup>

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* [22]) systems [1, 14, 44]. In reactive systems, the system interacts with the environment, and a correct

---

<sup>1</sup> To make life interesting, several different methodologies in system design are all termed “synthesis”. The automatic synthesis we study here 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.

system should then satisfy the specification with respect to all environments. If one applies the techniques of [18, 38] to reactive systems, one obtains systems that are correct only with respect to some environments. Pnueli and Rosner [44], Abadi, Lamport, and Wolper [1], and Dill [14] argued that 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. A correct system can be then viewed as a winning strategy in this game. It turns out that satisfiability of the specification is not sufficient to guarantee the existence of such a strategy. Abadi et al. called specifications for which a winning strategy exists *realizable*. Thus, 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 strategy can then be viewed as a labeling of a tree with directions in  $2^I$  by labels in  $2^O$ . The traditional algorithm for finding a winning strategy transforms the specification into a parity automaton over such trees. The automaton accepts a tree if the tree models a strategy all of whose computations satisfy the specification, and so a specification is realizable precisely when this tree automaton is nonempty, i.e., it accepts some infinite tree [44]. A finite generator of an infinite tree accepted by this automaton can be viewed as a finite-state system realizing the specification. This is closely related to the approach in [8, 47] to solve Church’s *solvability problem* [12]. In [32, 45, 54, 57] it was shown how this approach to system synthesis can be carried out in a variety of settings.

In spite of the rich theory developed for system synthesis, little of this theory has been reduced to practice. Some people argue that this is because the realizability problem, and hence also the synthesis problem, for linear-temporal logic (LTL) specifications is 2EXPTIME-complete [44, 49], but this argument is not compelling. First, experience with verification shows that even nonelementary algorithms can be practical, since the worst-case complexity does not arise often. For example, while the model-checking problem for specifications in second-order logic has nonelementary complexity, the model-checking tool MONA [17, 29] successfully verifies many specifications given in second-order logic. Furthermore, in some sense, synthesis is not harder than verification. This may seem to contradict the known fact that while verification is “easy” (linear in the size of the model and at most exponential in the size of the specification [35]), synthesis is hard (2EXPTIME-complete). There is, however, something misleading in this fact: while the complexity of synthesis is given with respect to the specification only, the complexity of verification is given with respect to the specification and the system, which can be much larger than the specification. In particular, it is shown in [49] that there are temporal specifications for which every realizing system must be at least doubly exponentially larger than the specifications. Clearly, the verification of such systems is doubly exponential in the specification, just as the cost of synthesis.

We believe that there are other reasons for the lack of practical impact of synthesis theory: algorithmic, methodological, scope, and qualitative. Let us start with the algorithmic difficulties. First, the construction of tree automata for realizing strategies uses determinization of Büchi automata. Safra’s determinization construction [51] has been notoriously resistant to efficient implementations [2, 53] (Safra’s construction was recently improved in [41]). While the new construction results in automata with fewer states, it suffers from the same problems that make Safra’s construction resistant to im-

plementations.) Second, determinization results in automata with a very complicated state space. The best-known algorithms for parity-tree-automata emptiness [26] are nontrivial already when applied to simple state spaces. Implementing them on top of the messy state space that results from determinization is awfully complex, and is not amenable to optimizations and a symbolic implementation.

Another major issue is methodological. The current theory of system synthesis assumes that one gets a comprehensive set of temporal assertions as a starting point. This cannot be realistic in practice. A more realistic approach would be to assume an *evolving* formal specification: temporal assertions can be added, deleted, or modified. Since it is rare to have a complete set of assertions at the very start of the design process, there is a need to develop *compositional* synthesis algorithms. Such algorithms can, for example, refine designs when provided with additional temporal properties. Moreover, development of complex systems is typically modular, with components being composed into larger components. Again, this is in contrast with current theory, which assumes a “flat” global specification. A more realistic approach would be to assume a *modular* setting, where specifications are given, and systems are generated, in a hierarchical manner.

A third drawback of current synthesis algorithms is their scope. Driven by the growing industrial impact of formal methods, various industrial efforts were launched recently to develop industrial-strength temporal assertion languages for semiconductor designs; e.g., Intel’s ForSpec and IBM’s Sugar [4, 6]. In the related application of model checking, theory has already bridged the gap with the new industrial-strength formalisms. In synthesis, theory still assumes specifications in traditional temporal logic like LTL. The richer formalisms require a nontrivial extension of current solutions. An even more interesting extension of the scope of synthesis refers to the underlying setting. It is by now realized that requiring the synthesized system to satisfy the specification against all possible environments is often too demanding. There is a need to define variants of synthesis that replace the universal quantification by a richer one.

Finally, while current automated synthesis algorithms generate a system that satisfies the specification, there is no emphasize on constructing optimal or well-structured systems. Indeed, the systems are obtained by extending algorithms that check nonemptiness of tree automata to return a witness to the nonemptiness, and there is no work on defining, measuring, and optimizing the quality of this witness. In particular, the synthesized systems performs as a black box, and there is no attempt to make its internal structure friendly to work with. Thus, automatic synthesis may result in systems that are larger and less structured than systems generated manually.

This paper surveys recent work that address the above challenges, describe new synthesis algorithms and paradigms, and discuss further challenges that they bring with them.

## 2 Preliminaries

Consider finite sets  $I$  and  $O$  of input and output signals, respectively. We model finite-state reactive systems with inputs in  $I$  and outputs in  $O$  by *transducers* ( $I/O$ -transducers, when  $I$  and  $O$  are not clear from the context). A transducer is a finite graph with a designated start state, where the edges are labeled by letters in  $2^I$  and the states are labeled

by letters in  $2^O$ . Formally, a transducer is a tuple  $\mathcal{T} = \langle I, O, S, s_{in}, \eta, L \rangle$ , where  $I$  and  $O$  are the sets of input and output signals,  $S$  is a finite set of states,  $s_{in} \in S$  is an initial state,  $\eta : S \times 2^I \rightarrow S$  is a deterministic transition function, and  $L : S \rightarrow 2^O$  is a labeling function. We extend  $\eta$  to words in  $(2^I)^*$  in the straightforward way. Thus,  $\eta : (2^I)^* \rightarrow S$  is such that  $\eta(\varepsilon) = s_{in}$ , and for  $x \in (2^I)^*$  and  $i \in 2^I$ , we have  $\eta(x \cdot i) = \eta(\eta(x), i)$ . Each transducer  $\mathcal{T}$  induces a *strategy*  $f_{\mathcal{T}} : (2^I)^* \rightarrow 2^O$  where for all  $w \in (2^I)^*$ , we have  $f_{\mathcal{T}}(w) = \tau(L(w))$ . Thus,  $f_{\mathcal{T}}(w)$  is the letter that  $\mathcal{T}$  outputs after reading the sequence  $w$  of input letters. The strategy  $f_{\mathcal{T}}$  generates computations over the set  $I \cup O$  of signals. A computation  $\rho \in (2^{I \cup O})^\omega$  is generated by  $f_{\mathcal{T}}$  if  $\rho = (i_0 \cup o_0), (i_1 \cup o_1), (i_2 \cup o_2), \dots$  and for all  $j \geq 1$ , we have  $o_j = f_{\mathcal{T}}(i_0 \cdot i_1 \cdots i_{j-1})$ . We sometimes refer to  $\rho$  as a computation of  $\mathcal{T}$ .

Linear temporal logic (LTL) is a formalism for specifying on-going behaviors of reactive systems [43]. Given an LTL formula  $\psi$  over the sets  $I$  and  $O$  of input and output signals, the *realizability problem* for  $\psi$  is to decide whether there is an  $I/O$ -transducer  $\mathcal{T}$  such that all the computations of  $\mathcal{T}$  satisfy  $\psi$  [44].

Algorithms for solving the synthesis problem are based on automata on infinite words and trees. We assume that the reader is familiar with the basic definitions of alternating tree automata. All the notations we are going to use are these defined and used in [33]. We do define here trees and labeled trees. Given a set  $D$  of directions, a  $D$ -tree is a set  $T \subseteq D^*$  such that if  $x \cdot c \in T$ , where  $x \in D^*$  and  $c \in D$ , then also  $x \in T$ . If  $T = D^*$ , we say that  $T$  is a full  $D$ -tree. The elements of  $T$  are called *nodes*, and the empty word  $\varepsilon$  is the *root* of  $T$ . For every  $x \in T$ , the nodes  $x \cdot c$ , for  $c \in D$ , are the *successors* of  $x$ . Given an alphabet  $\Sigma$ , a  $\Sigma$ -labeled  $D$ -tree is a pair  $\langle T, \tau \rangle$  where  $T$  is a tree and  $\tau : T \rightarrow \Sigma$  maps each node of  $T$  to a letter in  $\Sigma$ .

We denote classes of automata by acronyms in  $\{D, N\} \times \{B, C, P\} \times \{W, T\}$ . The first letter stands for the branching mode of the automaton (deterministic or nondeterministic); the second letter stands for the acceptance-condition type (Büchi, co-Büchi, or parity); the third letter stands for the object over which the automaton runs (words or trees). For example, NBW stands for nondeterministic Büchi automata, and DPT stands for deterministic parity tree automata.

### 3 Algorithms

The traditional algorithm for solving the realizability problem translates the LTL formula into an NBW, applies Safra's construction in order to get a DPW  $\mathcal{A}_\psi$  for it, expands  $\mathcal{A}_\psi$  to a DPT  $\mathcal{A}_{\forall\psi}$  that accepts all the trees all of whose branches satisfy  $\psi$ , and then checks the nonemptiness of  $\mathcal{A}_{\forall\psi}$  with respect to  $I$ -exhaustive  $2^{I \cup O}$ -labeled  $2^I$ -trees, namely  $2^{I \cup O}$ -labeled  $2^I$ -trees that contain, for each word  $w \in (2^I)^\omega$ , at least one path whose projection on  $2^I$  is  $w$  [44]. Thus, the algorithm applies Safra's determinization construction, and has to solve the nonemptiness problem for DPT. For  $\psi$  of length  $n$ , the DPW  $\mathcal{A}_\psi$  has  $2^{2^{O(n \log n)}}$  states and index  $2^{O(n)}$ . This is also the size of the DPT  $\mathcal{A}_{\forall\psi}$ , making the overall complexity doubly-exponential, which matches the lower bound in [49].

In [33], we describe a ‘‘Safraless’’ synthesis algorithm that avoids determinization and avoids the use of the parity acceptance condition. The algorithm proceeds as fol-

lows. A strategy  $f : (2^I)^* \rightarrow 2^O$  can be viewed as a  $2^O$ -labeled  $2^I$ -tree. We define a UCT  $\mathcal{S}_\psi$  such that  $\mathcal{S}_\psi$  accepts a  $2^O$ -labeled  $2^I$ -tree  $\langle T, \tau \rangle$  iff  $\tau$  is a good strategy for  $\psi$ . We define  $\mathcal{S}_\psi$  as follows.

Let  $\mathcal{A}_{\neg\psi} = \langle 2^{I \cup O}, Q, q_{in}, \delta, \alpha \rangle$  be an NBW for  $\neg\psi$  [55]. Thus,  $\mathcal{A}_{\neg\psi}$  accepts exactly all the words in  $(2^{I \cup O})^\omega$  that do not satisfy  $\psi$ . Then,  $\mathcal{S}_\psi = \langle 2^O, 2^I, Q, q_{in}, \delta', \alpha \rangle$ , where for every  $q \in Q$  and  $o \in 2^O$ , we have  $\delta'(q, o) = \bigwedge_{i \in 2^I} \bigwedge_{q' \in \delta(q, i \cup o)} (i, q')$ . Thus, from state  $q$ , reading the output assignment  $o \in 2^O$ , the automaton  $\mathcal{S}_\psi$  branches to each direction  $i \in 2^I$ , with all the states  $q'$  to which  $\delta$  branches when it reads  $i \cup o$  in state  $q$ . It is not hard to see that  $\mathcal{S}_\psi$  accepts a  $2^O$ -labeled  $2^I$ -tree  $\langle T, \tau \rangle$  iff for all the paths  $\{\varepsilon, i_0, i_0 \cdot i_1, i_0 \cdot i_1 \cdot i_2, \dots\}$  of  $T$ , the infinite word  $(i_0 \cup \tau(\varepsilon)), (i_1 \cup \tau(i_0)), (i_2 \cup \tau(i_0 \cdot i_1)), \dots$  is not accepted by  $\mathcal{A}_{\neg\psi}$ ; thus all the computations generated by  $\tau$  satisfy  $\psi$ . The size of  $\mathcal{A}_{\neg\psi}$  is exponential in the length of  $\psi$ . Using the rank-based method of [33], it is possible to reduce the nonemptiness of a UCT to the nonemptiness of an NBT with another exponential blow-up. Hence, realizability of  $\psi$  is reduced to checking the nonemptiness of an NBT of size doubly exponential in the length of  $\psi$ . Since the nonemptiness of an NBT can be solved in quadratic time, we are done. Moreover, as with the traditional algorithm [46], the approach in [33] can return a witness to the nonemptiness of  $\mathcal{S}_\psi$  – a transducer that realizes  $\psi$ . Thus we solve both realizability and synthesis.

The approach in [33] was improved in [31], where the algorithm starts by translating  $\neg\psi$  to a nondeterministic generalized Büchi automaton. The Safraless approach is used in algorithms for bounded synthesis [15, 52], was extended to timed specifications [21], led to further simplified synthesis algorithms [19], and was implemented in [25].

## 4 Methodology

A serious drawback of current synthesis algorithms is that they assume a comprehensive set of temporal assertions, describing the global behavior of the system, as a starting point. The ultimate goal in compositional synthesis is to compose a system that realizes a set of specifications from systems that realize the underlying specifications. Moreover, in case the conjunction of specifications is not realizable, the user can omit or weaken some of them.

In [31], we describe how it is possible, when we check the realizability of  $\psi \wedge \psi'$ , to use much of the work done in checking the realizability of  $\psi$  and  $\psi'$  in isolation. Recall that the Safraless algorithm reduces realizability of a specification  $\psi$  to the nonemptiness of an NBT  $\mathcal{U}_\psi$ . The state space of  $\mathcal{U}_\psi$  is simple: each state of  $\mathcal{U}_\psi$  is of the form  $\langle S, g \rangle$ , where  $S$  is a set of states in the intermediate UCT  $\mathcal{S}_\psi$  (the “state component”, which coincides with the set of states of  $\mathcal{A}_{\neg\psi}$ ) and  $g$  is a function that maps the states in  $S$  to a finite set of ranks (the “ranking component”). Realizability of  $\psi \wedge \psi'$  then involves the product of  $\mathcal{U}_\psi$  and  $\mathcal{U}_{\psi'}$ . The simple structure of the NBTs makes it possible not only to define the product easily (ease follows also from the use of a generalized Büchi acceptance condition), but also to use the work done during the nonemptiness checks of  $\mathcal{U}_\psi$  and  $\mathcal{U}_{\psi'}$  when we check the nonemptiness of the product. Indeed, the rank component of a state in the product describes ranks to states in both  $\mathcal{S}_\psi$  and  $\mathcal{S}_{\psi'}$ . A state whose projection on one of the components corresponds to a state that is empty

in the NBT for this component, can be labeled empty. This approach is especially helpful when combined with an incremental approach, where we construct the NBT with a small maximal rank, and increase the maximal rank only if the specification turns out not to be realizable with this small maximal rank,

So far, we considered compositionality in terms of the specification. In practice, the compositionality of the specification is often reflected also in a hierarchical structure, where subformulas of the specifications are composed not only in a Boolean manner but also in a modular one. Another way to view this is as follows. In the classical synthesis algorithms, it is always assumed the system is “constructed from scratch” rather than “composed” from reusable components. This rarely happens in real life. In real life, almost every non-trivial commercial system, either hardware or software, relies heavily on using libraries of reusable components. Furthermore, other contexts, such as web service orchestration, can be modeled as synthesis of a system from a library of components.

In [37], the authors define and study the problem of LTL synthesis from libraries of reusable components. Two notions of composition are defined: functional composition, for which the problem turns out not be decidable, and structural composition, for which the problem is 2EXPTIME-complete. As a side benefit, [37] derives an explicit characterization of the information needed by the synthesizer on the underlying components. This characterization can be used as a specification formalism between component providers and integrators. Synthesis from underlying components is extended in [36] to consider the problem of control-flow synthesis from libraries of probabilistic components. It is shown that this more general problem is also decidable.

## 5 Scope

One approach to tackle the algorithmic difficulties in LTL synthesis has been to restrict the class of allowed specification. In [5], the authors study the case where the LTL formulas are of the form  $\mathbf{G}p$ ,  $\mathbf{F}p$ ,  $\mathbf{GF}p$ , or  $\mathbf{FG}p$ .<sup>2</sup> In [3], the authors consider the fragment of LTL consisting of boolean combinations of formulas of the form  $\mathbf{G}p$ , as well as a richer fragment in which the  $\mathbf{N}$  operator is allowed. Since the games corresponding to formulas of these restricted fragments are simpler, the synthesis problem is simpler too, and it can be solved in PSPACE or EXPSPACE, depending on the specific fragment. Another fragment of LTL, termed  $GR(1)$ , is studied in [42]. In the  $GR(1)$  fragment (generalized reactivity(1)) the formulas are of the form  $(\mathbf{GF}p_1 \wedge \mathbf{GF}p_2 \wedge \dots \wedge \mathbf{GF}p_m) \rightarrow (\mathbf{GF}q_1 \wedge \mathbf{GF}q_2 \wedge \dots \wedge \mathbf{GF}q_n)$ , where each  $p_i$  and  $q_i$  is a Boolean combination of atomic propositions. It is shown in [42] that for this fragment, the synthesis problem can be solved in EXPTIME, and with only  $O((mn \cdot 2^{|AP|})^3)$  symbolic operations, where  $AP$  is the set of atomic propositions.

In [34], we continue this approach with the aim of focusing on properties that are used in practice. We study the synthesis problem for TRIGGER LOGIC. Modern industrial-strength property-specification languages such as Sugar [6], ForSpec [4], and the recent standards PSL [16] and SVA [56] include regular expressions. TRIGGER

<sup>2</sup> The setting in [5] is of real-time games, which generalizes synthesis.

LOGIC is a fragment of these logics that covers most of the properties used in practice by system designers. Technically, TRIGGER LOGIC consists of positive Boolean combinations of assertions about regular events, connected by the usual regular operators as well as temporal implication,  $\mapsto$  (“triggers”). For example, the TRIGGER LOGIC formula  $(true[*]; req; ack) \mapsto (true[*]; grant)$  holds in an infinite computation if every request that is immediately followed by an acknowledge is eventually followed by a grant. Also, the TRIGGER LOGIC formula  $(true[*]; err) \mapsto avoid(true[*]; ack)$  holds in a computation if once an error is detected, no acks can be sent.

It is shown in [34] that TRIGGER LOGIC formulas can be translated to deterministic Büchi automata using the two classical subset constructions: the determinization construction of [48] and the break-point construction of [39]. Accordingly, while the synthesis problem for TRIGGER LOGIC is still 2EXPTIME-complete, the synthesis algorithm for it is significantly simpler than the one used in general temporal synthesis.

The work described above stays in the traditional setting of synthesis and considers variants of the specification formalism. An even more interesting extension of the scope of synthesis refers to the underlying setting. It is by now realized that requiring the synthesized system to satisfy the specification against all possible environments is often too demanding. Dually, allowing all possible systems is perhaps not demanding enough. This issue is traditionally approached by adding assumptions on the system and/or the environment, which are modeled as part of the specification (c.f., [11]).

In [30, 52], the authors study *bounded temporal synthesis*, in which bounds on the sizes of the state space of the system and the environment are additional parameters to the synthesis problem. The study is motivated by the fact that such bounds may indeed change the answer to the synthesis problem, as well as the theoretical and computational aspects of the synthesis problem. In particular, a finer analysis of synthesis, which takes system and environment sizes into account, yields deeper insight into the quantificational structure of the synthesis problem and the relationship between strong synthesis – there exists a system such that for all environments, the specification holds, and weak synthesis – for all environments there exists a system such that the specification holds.

Unlike the unbounded setting, where determinacy of regular games implies that strong and weak synthesis coincide, these notions do not coincide in the bounded setting. Bounding the size of the system or the environment turns the synthesis problem into a search problem, and one cannot expect to do better than brute-force search. In particular, the synthesis problem for bounded environments is NP-complete [52], and is  $\Sigma_2^P$ -complete for bonded systems and environments (the complexity is in terms of the bounds, for a specification given by a deterministic automaton) [30].

A different, more conceptual change of the setting has to do with the fact that modern systems often interact with other systems. For example, the clients interacting with a server are by themselves distinct entities (which we call agents) and are many times implemented by systems. In the traditional approach to synthesis, the way in which the environment is composed of its underlying agents is abstracted. In particular, the agents can be seen as if their only objective is to conspire to fail the system. Hence the term “hostile environment” that is traditionally used in the context of synthesis. In real life, however, many times agents have goals of their own, other than to fail the system. The

approach taken in the field of algorithmic game theory [40] is to assume that agents interacting with a computational system are *rational*, i.e., agents act to achieve their own goals. Assuming agents rationality is a restriction on the agents behavior and is therefore equivalent to restricting the universal quantification on the environment.

In [20], we introduce the problem of synthesis in the context of rational agents (*rational synthesis*, for short). The input consists of a temporal-logic formula specifying the system, temporal-logic formulas specifying the objectives of the agents, and a solution concept definition. The output is an implementation  $T$  of the system and a profile of strategies, suggesting a behavior for each of the agents. The output should satisfy two conditions. First, the composition of  $T$  with the strategy profile should satisfy the specification. Second, the strategy profile should be an equilibrium in the sense that, in view of their objectives, agents have no incentive to deviate from the strategies assigned to them, where “no incentive to deviate” is interpreted as dictated by the given solution concept. As it turns out, system synthesizers can capitalize on the rationality and goals of the agents interacting with it. Moreover, for the classical definitions of equilibria studied in game theory, rational synthesis is not harder than traditional synthesis. The results in [20] also consider the multi-valued case in which the objectives of the system and the agents are still temporal logic formulas, but involve payoffs from a finite lattice.

## 6 Quality

Very little attention has been paid to the quality of the systems that are automatically synthesized. Typically, many systems satisfy a realizable specification, and while they all satisfy the specification, they may do so at different levels of “unspecified quality”. This latter problem is a real obstacle, as designers would be willing to give up manual design only after being convinced that the automatic procedure that replaces it generates systems of comparable quality. Nowadays specification formalisms are too abstract to specify such quality measures.

An approach in which quantitative reasoning is used in order to improve the quality of automatically synthesized systems is described in [7]. There, the synthesis problem is reduced to the solution of lexicographic mean-payoff games. A winning strategy in the game induces a system that satisfies the specification in high quality. Another approach is described in [27, 28], where the authors introduce the idea of model-checking-based genetic programming as a general approach to synthesis: starting with a randomly generated solution, the solution is iteratively improved according to a valuation (fitness function) that a model checker assigns to the suggested solution.

The neglecting of the quality of automatically synthesized systems is related to the fact that model checking is Boolean. The Boolean fate of a verification process seems natural, as a system can either satisfy its specification or not satisfy it. The richness of today’s systems and verification methodologies has motivated the introduction of *multi-valued* specification formalisms. These formalisms are used in order to specify quantitative properties (say, map a computation to the maximal wait time along it) [9, 10, 23], or in order to specify rich truth values (say, an “unknown” value, in the case of abstraction [50], or a value that is a subset of all possible viewpoints, in case of systems with multiple viewpoints [24]). Very little attempts, however, have been made to

augment temporal logics with a quantitative layer that would enable the specification of the relative merits of different aspects of the specification. The idea behind such a layer is that there should be a difference between a system that satisfies the specification  $\mathbf{AG}(req \rightarrow \mathbf{F}grant)$  with grants generated soon after requests, and a system that satisfies it with grants generated after long waits. Specification formalisms should be refined in order to reveal such differences, and algorithms for reasoning about the new formalisms are required. With the right formalisms, a designer will be able to associate the different aspects of the specification with costs and rewards, reflecting their importance, and synthesis algorithms will be able to generate systems that not only satisfy the specification, but also do so in a way that maximizes the quality of the satisfaction, as defined formally by the designer. The development of such multi-valued formalisms and algorithms for reasoning about them involves is the subject of current research.

## References

1. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 25th Int. Colloq. on Automata, Languages, and Programming*, LNCS 372, pages 1–17. Springer, 1989.
2. C.S. Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of Büchi automata. In *Proc. 10th Conf. on the Implementation and Application of Automata*, LNCS 3845, pages 262–272. Springer, 2005.
3. R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. *ACM Transactions on Computational Logic*, 5(1):1–25, 2004.
4. R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M.Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification logic. In *Proc. 8th Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2280, pages 196–211. Springer, 2002.
5. E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier, 1998.
6. I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh. The temporal logic Sugar. In *Proc. 13th Conf. on Computer Aided Verification*, LNCS 2102, pages 363–367. Springer, 2001.
7. R. Bloem, K. Chatterjee, T. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *Proc. 21st Conf. on Computer Aided Verification*, LNCS 5643, pages 140–156. Springer, 2009.
8. J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138:295–311, 1969.
9. A. Chakrabarti, K. Chatterjee, T.A. Henzinger, O. Kupferman, and R. Majumdar. Verifying quantitative properties using bound functions. In *Proc. 13th Conf. on Correct Hardware Design and Verification Methods*, LNCS 3725, pages 50–64. Springer, 2005.
10. K. Chatterjee, L. Doyen, and T. Henzinger. Quantitative languages. In *Proc. 17th Annual Conf. of the European Association for Computer Science Logic*, pages 385–400, 2008.
11. K. Chatterjee, T. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *Proc. 19th Conf. on Concurrency Theory*, LNCS 5201, pages 147–161. Springer, 2008.
12. A. Church. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35. Institut Mittag-Leffler, 1963.
13. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
14. D.L. Dill. *Trace theory for automatic hierarchical verification of speed independent circuits*. MIT Press, 1989.

15. R. Ehlers. Symbolic bounded synthesis. In *Proc. 22nd Conf. on Computer Aided Verification*, LNCS 6174, pages 365–379. Springer, 2010.
16. C. Eisner and D. Fisman. *A Practical Introduction to PSL*. Springer, 2006.
17. J. Elgaard, N. Klarlund, and A. Möller. Mona 1.x: new techniques for WS1S and WS2S. In *Proc. 10th Conf. on Computer Aided Verification*, LNCS 1427, pages 516–520. Springer, 1998.
18. E.A. Emerson and E.M. Clarke. Using branching time logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.
19. E. Filiot, N. Jin, and J.-F. Raskin. An antichain algorithm for LTL realizability. In *Proc. 21st Conf. on Computer Aided Verification*, LNCS 5643, pages 263–277, 2009.
20. D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *Proc. 16th Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 6015, pages 190–204. Springer, 2010.
21. B. Di Giampaolo, G. Geeraerts, J.-F. Raskin, and N. Sznajder. Safrless procedures for timed specifications. In *Proc. 8th international conference on Formal modeling and analysis of timed systems*, LNCS 6246, pages 2–22. Springer, 2010.
22. D. Harel and A. Pnueli. On the development of reactive systems. In K. Apt, editor, *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, pages 477–498. Springer, 1985.
23. T.A. Henzinger. From Boolean to quantitative notions of correctness. In *Proc. 37th ACM Symp. on Principles of Programming Languages*, pages 157–158, 2010.
24. A. Hussain and M. Huth. On model checking multiple hybrid views. Technical Report TR-2004-6, University of Cyprus, 2004.
25. B. Jobstmann and R. Bloem. Game-based and simulation-based improvements for LTL synthesis. In *3rd Workshop on Games in Design and Verification*, 2006.
26. M. Jurdzinski. Small progress measures for solving parity games. In *Proc. 17th Symp. on Theoretical Aspects of Computer Science*, LNCS 1770, pages 290–301. Springer, 2000.
27. G. Katz and D. Peled. Genetic programming and model checking: Synthesizing new mutual exclusion algorithms. In *6th Symp. on Automated Technology for Verification and Analysis*, LNCS 5311, pages 33–47. Springer, 2008.
28. G. Katz and D. Peled. Model checking-based genetic programming with an application to mutual exclusion. In *Proc. 14th Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 4963, pages 141–156. Springer, 2008.
29. N. Klarlund. Mona & Fido: The logic-automaton connection in practice. In *Proc. 6th Annual Conf. of the European Association for Computer Science Logic*, Lecture Notes in Computer Science, 1998.
30. O. Kupferman, Y. Lustig, M.Y. Vardi, and M. Yannakakis. Temporal synthesis for bounded systems and environments. In *Proc. 28th Symp. on Theoretical Aspects of Computer Science*, pages 615–626, 2011.
31. O. Kupferman, N. Piterman, and M.Y. Vardi. Safrless compositional synthesis. In *Proc. 18th Conf. on Computer Aided Verification*, LNCS 4144, pages 31–44. Springer, 2006.
32. O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In *Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.
33. O. Kupferman and M.Y. Vardi. Safrless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.
34. O. Kupferman and M.Y. Vardi. Synthesis of trigger properties. In *Proc. 16th Conf. on Logic for Programming Artificial Intelligence and Reasoning*, LNCS 6355, pages 312–331. Springer, 2010.
35. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, 1985.

36. Y. Lustig, S. Nain, and M.Y. Vardi. Synthesis from probabilistic components. In *Proc. 20th Annual Conf. of the European Association for Computer Science Logic*, pages 412–427, 2011.
37. Y. Lustig and M.Y. Vardi. Synthesis from components. In *Proc. 12th Conf. on Foundations of Software Science and Computation Structures*, LNCS 5504, pages 395–409. Springer, 2009.
38. Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93, 1984.
39. S. Miyano and T. Hayashi. Alternating finite automata on  $\omega$ -words. *Theoretical Computer Science*, 32:321–330, 1984.
40. N. Nisan, T. Roughgarden, E. Tardos, and V.V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.
41. N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 255–264. IEEE press, 2006.
42. N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive(1) designs. In *Proc. 7th Conf. on Verification, Model Checking, and Abstract Interpretation*, LNCS 3855, pages 364–380. Springer, 2006.
43. A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
44. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.
45. A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proc. 16th Int. Colloq. on Automata, Languages, and Programming*, LNCS 372, pages 652–671. Springer, 1989.
46. M.O. Rabin. Weakly definable relations and special automata. In *Proc. Symp. Math. Logic and Foundations of Set Theory*, pages 1–23. North Holland, 1970.
47. M.O. Rabin. Automata on infinite objects and Church’s problem. *Amer. Mathematical Society*, 1972.
48. M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.
49. R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, 1992.
50. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *Proc. 9th Conf. on Computer Aided Verification*, LNCS 1254, pages 72–83. Springer, 1997.
51. S. Safra. On the complexity of  $\omega$ -automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, 1988.
52. S. Schewe and B. Finkbeiner. Bounded synthesis. In *5th Symp. on Automated Technology for Verification and Analysis*, LNCS 4762, pages 474–488. Springer, 2007.
53. S. Tasiran, R. Hojati, and R.K. Brayton. Language containment using non-deterministic omega-automata. In *Proc. 8th Conf. on Correct Hardware Design and Verification Methods*, LNCS 987, pages 261–277. Springer, 1995.
54. M.Y. Vardi. An automata-theoretic approach to fair realizability and synthesis. In *Proc. 7th Conf. on Computer Aided Verification*, LNCS 939, pages 267–292. Springer, 1995.
55. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.
56. S. Vijayaraghavan and M. Ramanathan. *A Practical Guide for SystemVerilog Assertions*. springer, 2005.
57. H. Wong-Toi and D.L. Dill. Synthesizing processes and schedulers from temporal specifications. In *Proc. 2nd Conf. on Computer Aided Verification*, volume 3 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 177–186. AMS, 1991.