Experimental Aspects of Synthesis

Rüdiger Ehlers
Reactive systems group, Saarland University

We discuss the problem of experimentally evaluating linear-time temporal logic (LTL) synthesis tools for reactive systems. We first survey previous such work for the currently publicly available synthesis tools, and then draw conclusions by deriving useful schemes for future such evaluations.

In particular, we explain why previous tools have incompatible scopes and semantics and provide a framework that reduces the impact of this problem for future experimental comparisons of such tools. Furthermore, we discuss which difficulties the complex workflows that begin to appear in modern synthesis tools induce on experimental evaluations and give answers to the question how convincing such evaluations can still be performed in such a setting.

1 Introduction

The problem of synthesizing reactive systems from linear-time temporal logic (LTL) specifications [58, 1, 59, 46, 73] has attracted many researchers in the past, leading to a tremendous amount of results in this area. Broadly, works contributing to the progress of its solution can be classified into two sorts.

On the theory side, major breakthroughs have been obtained by establishing the 2EXPTIME-hardness of this problem [60, 74, 25], along with asymptotically optimal automata-theoretic constructions for solving it [59, 1, 73]. Recent works are concerned with making these constructions easier [49, 65, 57] or enhancing the scope of the algorithms and hardness-results known to, e.g., distributed systems [48, 24].

On the practical side, many works deal with the construction of sophisticated algorithms that aim at improving the scalability of current synthesis techniques [38, 66, 22, 23, 57, 16]. While the 2EXPTIME-hardness of the LTL synthesis problem induces a limit on the effectiveness of any approach, this line of research is motivated by the observation that "typical" specifications in practice have a structure that can be exploited [44, 67, 52, 23, 66, 57, 16, 45].

While many works fall into both categories, contributions to the latter sort typically contain proofs of the usefulness of the proposed techniques obtained by experimentally evaluating a prototype implementation. This is commonly done by taking some example specifications (the so-called *benchmarks*) and showing that the prototype is able to handle these in reasonable time.

The situation is similar to the one for the problem of *satisfiability* (SAT) testing (see, e.g., [4]), where despite its NP-completeness, an active area of research has emerged, witnessing its progress by the fact that practical problems with millions of variables are nowadays routinely solved by modern SAT solvers. One of the key factors for this success is the possibility to perform meaningful benchmarking, which drives the development of new solution heuristics into those directions that appear to be most promising. Thousands of example problem instances can easily be used to obtain, optimise and test new approaches. As a result, the annual SAT solving competitions typically draw a lot of interest, for example 19 submissions to the main track in the 2010 SAT-Race [55].

For the synthesis of reactive systems from linear-time specifications, however, there appears to be far less interest in tools. At the time of writing, to the best of our knowledge, there are only four publicly

Johannes Reich and Bernd Finkbeiner (Eds): International Workshop on Interactions, Games and Protocols (iWIGP)
EPTCS 50, 2011, pp. 1–16, doi:10.4204/EPTCS.50.1

available synthesis tools[1], namely ANZU [9, 39, 7, 8], LILY [37, 38], ACACIA [13, 22, 23], and UNBEAST [17, 16]. Equally unfortunate, the amount of benchmarks available is rather low. One can identify (at least) three reasons for this state:

1. A factor contributing to this difference is the fact that while rudimentary SAT solvers can be written in the order of hours, creating even a simple synthesis tool requires the implementation of comparably complex operations. As an example, for approaches working with deterministic automata, a construction similar to Safra's determinisation procedure [64] needs to be performed, which has been argued to be notoriously complex to implement [35, 34, 38, 44, 45].
2. As a second reason, the publication schemes of the formal methods and SAT communities are different. While the latter appreciates work whose primary concern is to improve the scalability of current SAT solving techniques (see, e.g., [42, 54, 28] for recent such publications from 2010), works in the formal methods area are typically built around some appealing new concept, which is mostly only evaluated briefly on some prototype implementation (see, e.g., [29] for a pointer to a typical such a case in the area of software model checking, or simply compare recent practical synthesis papers [23, 16, 22, 38, 57, 66, 67]).

   Arguably, one of the main reasons for this difference is the fact that the roots of formal methods lie in theoretical computer science, where, given technical correctness and sufficient general style of writing, the main merits of a paper are seen in the significance of the conceptual contribution to the field [56]. As a result, papers that propose improvements to current techniques that lack a major theoretical insight have a small chance of being accepted at major conferences even if the proposed techniques lead to significant speed-ups in the synthesis process. Consequently, time is typically only invested in writing synthesis tools after a new idea has been developed that is *both* theoretically compelling and gives the impression that it will significantly improve upon the performance of previous techniques.
3. Third, even if time has been invested in writing a synthesis tool, a new technique still has to be shown to be competitive with earlier techniques. Typically, benchmarking is used for this purpose. In the scope of synthesis, however, it can be observed that this is by no means a trivial task – all four currently available synthesis tools have different scopes and semantics. Also, the amount of benchmarks available is extremely low and the benchmarks in previous evaluations have often been rewritten to be compatible to the improvements proposed. This puts a high burden on the quality of future works in this area: comparability to previous works must be maintained in order to obtain a high credibility of the experimental evaluation. At the same time, as improvements to synthesis techniques often introduce additional details that must be taken care of in the evaluation (e.g., having two semi-algorithms running in parallel in [22, 16] or the assumption dropping heuristic from [23]), it is very hard to produce an appealing evaluation without spending too much space in a publication on the details.

Recently, the first of these problems has been attenuated by the availability of good LTL-to-Büchi translation tools [63, 27, 14] and determinisation and optimisation tools for ω-automata [40, 15, 18]

[1] For the scope of this paper, we exclude all tools that aim at pure game solving, as here, the synthesis functionalities and the possibility to start from linear-time temporal logic is missing (which is typically not merely a preprocessing step). There is another tool called RATSY [6], which we excluded as its synthesis functionality is mainly a reinterpretation of ANZU (plus some preliminary implementation of the bounded synthesis approach [65, 22]) and the tool aims at providing an environment for specification engineering rather than being only a synthesis tool. Consequently, no experimental evaluation of the synthesis performance has been given in [6]. The JTLV [61] scripting environment that also has synthesis procedures has been excluded as no stand-alone synthesis tool exists and benchmarks comparisons are not available.

on the one hand, and the availability of efficient binary decision diagram (BDD) libraries [68, 10] and satisfiability modulo theory (SMT) or SAT-solvers [4] as reasoning backbones on the other hand. Thus, developers of new synthesis tools can build their implementations on top of such previous work. The second problem will hopefully lose impact over time when more interest in the practical side of reactive system synthesis is aroused.

In this paper, we approach the remaining third problem by giving both insights why experimental evaluations in the context of synthesis are notoriously harder than, e.g., in the SAT context, as well as proposing "standardised" evaluation schemes for synthesis tools that aim at simplifying further work in this area. We hope that our discussion helps interested observers of the advances in the practical approaches to synthesis (by providing a survey on the problem of evaluating synthesis tools, with special consideration of work already done in this area) as well as authors of future synthesis tools (by giving inspirations and in particular justification for the choice of their experimental settings) and paper reviewers in the field (by explaining the difficulties of performing an experimental evaluation for synthesis tools).

We start by giving a definition of the LTL (open) synthesis problem in Section 2. In Section 3, we review the synthesis approaches of the synthesis tools that were publicly available at the time of writing. Afterwards, we give some observations on these approaches (and their experimental evaluations). In Section 5, we analyse the observations and propose a framework for future synthesis tool experimental evaluations. We conclude with a summary.

2 The LTL open synthesis problem

We start by giving a problem description of reactive system synthesis that focusses on those aspects that require special attention when comparing synthesis approaches. Formally, a synthesis problem instance is a triple $\langle \mathsf{AP}_I, \mathsf{AP}_O, \psi \rangle$, where AP_I is a set of input atomic propositions, AP_O is a set of output atomic propositions, and ψ is a formula in linear-time temporal logic (LTL) [58] over $\mathsf{AP}_I \uplus \mathsf{AP}_O$. For the scope of this paper, we denote the LTL temporal operators for "globally", "finally" and "next-time" by G, F, and X. For the ease of reading, we sometimes call the atomic propositions simply variables or bits.

We say that a triple $\langle \mathsf{AP}_I, \mathsf{AP}_O, \psi \rangle$ represents a *realisable* specification in the *Mealy-type semantics* if there exists some function $f : (2^{\mathsf{AP}_I})^+ \to 2^{\mathsf{AP}_O}$ such that for all $w = w_0 w_1 \ldots \in \mathsf{AP}_I^\omega$, we have $(w_0 \cup f(w_0)), (w_1 \cup f(w_0 w_1)), (w_2 \cup f(w_0 w_1 w_2)), \ldots \models \psi$. Likewise, we say that $\langle \mathsf{AP}_I, \mathsf{AP}_O, \psi \rangle$ is realisable in the *Moore-type semantics* if there exists some function $f : (2^{\mathsf{AP}_I})^* \to 2^{\mathsf{AP}_O}$ such that for all $w = w_0 w_1 \ldots \in \mathsf{AP}_I^\omega$, we have $(w_0 \cup f(\varepsilon)), (w_1 \cup f(w_0)), (w_2 \cup f(w_0 w_1)), \ldots \models \psi$ for ε denoting the empty word. Specifications that are not realisable are called unrealisable in the respective semantics.

Typically, realisability checking is performed by building a game between a *system player* and an *environment player*. In this setting, a function f satisfying the constraints stated above is called a *winning strategy*. Details on the game-based view to synthesis can be found in [32].

It is well-known that whenever there exists some winning strategy for one of the semantics above, there also exists a finite representation of it. For the Mealy-type semantics, this representation is typically given as a Mealy automaton, whereas for the Moore-type semantics, Moore automata serve this purpose (see, e.g., [53]).

Intuitively, the Mealy- and Moore-type semantics differ in the order of input and output. As an example, for the LTL formula $\psi = \mathsf{G}(r \leftrightarrow g)$, the specification $\langle \{r\}, \{g\}, \psi \rangle$ is realisable for the Mealy-type semantics but not for the Moore-type semantics. The reason is that in the Mealy-type semantics, the system already knows the input in the respective computation cycle when having to choose an output, whereas in the Moore-type semantics, the roles are swapped and thus the system has to guess whether r

is set or not when choosing whether g should be set.

3 Synthesis tools

We briefly recapitulate the ideas behind the synthesis tools ANZU [39], LILY [37, 38], ACACIA [13, 22, 23], and UNBEAST [17, 16]. We use the terminologies from the respective papers and refer the reader not familiar with the terms used hereafter to them.

3.1 ANZU

Scope: This tool implements the concept of generalised reactivity(1) [57] synthesis (abbreviated as GR(1) synthesis) in the Mealy-type semantics. Here, the specification is restricted to be of the form $(a_1 \wedge a_2 \wedge \ldots \wedge a_n) \rightarrow (g_1 \wedge g_2 \wedge \ldots \wedge g_m)$ for some sets of assumptions $\{a_1, \ldots, a_n\}$ and guarantees $\{g_1, \ldots, g_m\}$. Every assumption is of one of the following forms:

(1) ψ_I

(2) $\mathsf{G}(\psi \rightarrow \mathsf{X}(\psi_I))$

(3) $\mathsf{GF}(\psi)$

where ψ is an LTL-formula over $\mathsf{AP}_I \cup \mathsf{AP}_O$ free of temporal operators and ψ_I is an LTL-formula over AP_I free of temporal operators . Likewise, all guarantees are of one of the following forms:

(1) ψ_O

(2) $\mathsf{G}(\psi \rightarrow \mathsf{X}(\psi_O))$

(3) $\mathsf{GF}(\psi)$

where ψ_O is an LTL-formula over AP_O free of temporal operators. ANZU requires the assumptions and guarantees to be in *Property Specification Language* (PSL) [19] syntax and checks for *strong realisability* of the given overall specification, where unlike in normal realisability checking, safety guarantee violations are not tolerated in cases in which a safety assumption violation has not yet been witnessed but the system has a strategy to ensure that the overall input and output will not satisfy all assumptions (see, e.g., [41]). Specifications of the $(\bigwedge \text{assumptions}) \rightarrow (\bigwedge \text{guarantees})$ form, as used in ANZU, typically occur in cases in which a part of a larger system is to be synthesized, where the set of assumptions represents the behaviour the part of the system to be synthesized can assume about the other parts of the system whereas the guarantees describe the requirements on the behaviour of the part of the system to be synthesized.

Techniques: ANZU implements generalised reactivity(1) synthesis [57] in a symbolic manner using binary decision diagrams (BDDs) [10, 68] as reasoning backbone. Here, a synthesis game is built whose state space consists of all variable valuations to the input and output atomic propositions. The LTL assumptions and guarantees of the forms ψ_I and ψ_O are encoded into the set of initial positions of the game, whereas assumptions and guarantees of the form $\mathsf{G}(\psi \rightarrow \mathsf{X}(\psi_I))$ and $\mathsf{G}(\psi \rightarrow \mathsf{X}(\psi_O))$ are encoded into its transition relation (describing the possible moves of the players). Then, a symbolic algorithm is used in which the system player tries to satisfy the specification $(a'_1 \wedge a'_2 \wedge \ldots \wedge a'_{n'}) \rightarrow (g'_1 \wedge g'_2 \wedge \ldots \wedge g'_{m'})$ in this so-called *game arena* (consisting of the game positions and the transition relation), where $\{a'_1, \ldots, a'_{n'}\}$ are the assumptions of the form $\mathsf{GF}(\psi)$ and $\{g'_1, \ldots, g'_{m'}\}$ are the guarantees of the form $\mathsf{GF}(\psi)$. Implementations are extracted by building circuits for computing the outputs from

the BDD representation of the winning state set while performing a care-set optimisation to the BDD after every step [7, 8].

Experimental evaluation: The usefulness of the tool ANZU has been shown on two case studies: an AMBA AHB arbiter [2] specification (simplified by leaving out bus splits and early burst terminations) and a generalised Buffer (GenBuf) controller that has been described by IBM for tutorial purposes [36]. Both case studies contain several assumptions and guarantees and are scalable by the numbers of clients.

3.2 LILY

Scope: The tool LILY accepts arbitrary LTL formulas in PSL syntax as specifications. If multiple LTL formulas are found in the input file, they are treated in a conjunctive manner, except for those that are preceded by an `assume` keyword, which are used as assumptions for the overall specification.

Techniques: LILY implements the optimisations to the Safraless synthesis [49] approach presented in [38]. In the first step, the negation of the specification is converted to a node-labelled nondeterministic Büchi word (NBW) automaton using the LTL-to-Büchi translator WRING [69]. The Büchi automaton is then converted to a universal co-Büchi tree automaton (UCT) that checks for the satisfaction of the original specification along all computation tree paths, which is in turn tested for emptiness using a construction proposed by Kupferman and Vardi [49], utilising alternating weak tree automata (AWT) and non-deterministic Büchi tree (NBT) automata. Jobstmann and Bloem [38] add various optimisations to these steps. The conversion of the UCT to the AWT is parametrised by some constant k, which influences the size of the NBT produced and thus the running time of the overall algorithm. While low values for k typically suffice for realisable specifications in practice, a relatively large value of k, exponential in the size of the NBW (or, alternatively, doubly-exponential in the length of the LTL specification), is needed to have the emptiness of the language of the NBT imply the emptiness of the UCT language (except if the UCT turns out to be weak), and thus prove unrealisability of the given specification. To avoid this problem, LILY can also be run in a special unrealisability detection mode. In this case, it checks the realisability of the negated specification with swapped inputs and outputs (and a slight modification of the resulting specification to convert its Mealy-type semantics to Moore-type again). Then, for unrealisable specifications, only a small value of k is typically needed in practice to identify them as such.

Experimental evaluation: In [38], the performance of LILY is evaluated on some specifications written by the authors of that paper, representing mostly arbiter variations and traffic light controllers. The evaluation is focussed on proving that the optimisations proposed in that paper contribute significantly to having low running times of the tool.

3.3 ACACIA

Scope: The tool ACACIA has the same input syntax as LILY and also uses Moore-type semantics. There exist two versions of ACACIA. ACACIA 2009 implements the techniques described in [22], while ACACIA 2010 also implements those of [23]. The latter version includes support for making assumptions local to some set of guarantees. The specification then consists of a conjunction of sub-specifications of the form $(\bigwedge \text{assumptions}) \rightarrow (\bigwedge \text{guarantees})$. All assumptions and guarantees of the conjuncts are assumed to be given separately.

Techniques: ACACIA is based on the concept of bounded synthesis [65, 22], a refinement of the Safra-less synthesis techniques proposed in [49]. Here, as in LILY, the specification is first negated and then converted to a node-labelled Büchi automaton. As LILY, ACACIA uses WRING [69] for this purpose. Afterwards, the Büchi automaton is converted to a universal co-Büchi tree automaton (UCT) that checks for the satisfaction of the specification along all paths of a computation tree. This UCT is used as a basis for building a series of synthesis safety games, where for a successively increasing *bound value k*, for every state q in the UCT, the maximum number of visits to rejecting states in the UCT from its initial state along some path to q for the input/output played in the game so far is encoded into the game positions. Once one of these counters exceeds the value k, the game is lost for the system player. The main idea of ACACIA is to use anti-chains as an efficient representation of the *frontier sets* (i.e., pre-fixed points of winning positions) occurring during the safety game solving process. This representation makes use of the fact that the set of possible future behaviours of the system player in a position p_1 can only be larger than when being in a game position p_2 if all counters in p_1 are less than or equal to those in p_2. Thus by storing only states whose counter vectors are not *dominated* by the counter vectors of other states in the pre-fixed point during the solving process, redundancies can be avoided.

As in LILY, the value of k required to conclude the unrealisability of a specification is exponential in the size of the UCT or doubly-exponential in the length of the LTL specification. The authors thus propose to run ACACIA two times in parallel, where in the first run, realisability is checked and in the second run, unrealisability is tested. As in LILY, in the latter case, the specification is negated, a conversion between Mealy-type and Moore-type semantics takes place, and the inputs and outputs are swapped.

ACACIA2010 adds some additional features. Here, the game solving process is made compositional. Recall that in ACACIA 2010, the specification is supposed to consist of a conjunction of sub-specifications of the form $(\bigwedge \text{assumptions}) \rightarrow (\bigwedge \text{guarantees})$. In this setting, the safety games for the synthesis process can be built separately, preliminarily solved independently and finally composed on-the-fly during the solving process for the game representing the overall specification. ACACIA 2010 furthermore adds the possibility to use the OTFUR mixed forward-backward game solving algorithm [50, 11] instead of the classical backward safety game solving algorithm. Finally, for cases in which the specification is only a single formula of the form $(\bigwedge \text{assumptions}) \rightarrow (\bigwedge \text{guarantees})$, ACACIA can rewrite this specification into the form $\bigwedge_{g \in \text{guarantees}} ((\bigwedge \text{assumptions}) \rightarrow g)$ in order to benefit from the compositional algorithms implemented. In this case, an assumption dropping heuristic is used to remove some assumption copies in this formula, which reduces the problem that the assumptions are replicated for all guarantees in this setting. Using the heuristic makes the approach however incomplete.

Experimental evaluation: In [22], the focus of the experimental evaluation lies on proving that ACACIA improves upon the performance of LILY, using the fact that the semantics are compatible. The authors of [22] show that on the examples from [38], using the anti-chains approach typically results in lower computation times and that the Büchi automaton building time surprisingly dominates the overall synthesis time. One of the example specifications is made scalable and it is proven that the anti-chains approach is much faster here. Another set of variations of one of the LILY examples is used as a further benchmark set.

In [23], the 2010 version of ACACIA is evaluated with several different choices for (1) whether game solving should be performed backwards or in a forward-backward manner, (2) whether monolithic or compositional synthesis should be performed, and (3) whether the assumption dropping heuristic should be used (only in the compositional case). Apart from the benchmarks also used in [22], the

generalised Buffer (GenBuf) controller [36] specification that was also used for benchmarking ANZU has been formulated in a way such that the assumptions to the environment are local to some sets of guarantees, such that compositional synthesis can be performed directly. This benchmark is used to show the benefits of the compositional approach.

3.4 UNBEAST

Scope: The tool UNBEAST [17, 16] focusses on specifications of the form $(\bigwedge \text{assumptions}) \rightarrow (\bigwedge \text{guarantees})$ and uses Mealy-type semantics. By using an input language based on XML, incorrect presumptions by the user about precedences of temporal operators in LTL are avoided. The assumptions and guarantees are given separately in the XML input file.

Techniques: The UNBEAST tool implements the synthesis techniques presented in [65, 16]. The library CUDD [68] is used for constructing and manipulating BDDs during the synthesis process.

The first step is to determine which of the given assumptions and guarantees are safety formulas. In order to detect also simple cases of *pathological safety* [47], this is done by computing an equivalent Büchi automaton using an external LTL-to-Büchi converter such as LTL2BA [27] or SPOT's LTL2TGBA [14], and examining whether all maximal strongly connected components in the computed automaton do not have infinite non-accepting paths. Special care is taken of so-called of *bounded look-ahead safety formulas*.

In a second step, for the set of bounded look-ahead assumptions and the set of such guarantees, safety automata for their respective conjunctions are built. Both of them are represented in a symbolic way using BDDs. For the remaining safety assumptions and guarantees, safety automata are built by taking the Büchi automata computed in the previous step and applying a subset construction for determinisation in a symbolic manner. For the remaining non-safety parts of the specification, a combined universal co-Büchi automaton is computed by calling the external LTL-to-Büchi tool again.

In the next phase, the given specification is checked for realisability. This is done almost as in ACACIA 2009, i.e., for a successively increasing so-called *bound value*, the bounded synthesis approach [65, 22] is performed by building a safety automaton from the co-Büchi automaton for the non-safety part of the specification and solving the safety games induced by a special product of the automata involved [16]. However, instead of anti-chains, BDDs are used.

Finally, if the specification is found to be realisable (i.e., the game computed in the previous phase is winning for the player representing the system to be synthesised), the symbolic representation of the winning states of the system is used to compute a prototype implementation satisfying the specification in a fully symbolic way, using a slight simplification of the algorithm from [43]. However, the implementations generated are typically relatively large. As ACACIA, to also detect unrealisable specifications, UNBEAST needs to be run two times in parallel.

Experimental evaluation: UNBEAST was evaluated on the specifications defined in [38] as well as on those given in [22]. The Moore-type semantics from these examples have been adapted to the Mealy-type semantics of UNBEAST by prefixing all occurrences of input atomic propositions with an LTL next-time operator (see, e.g., [38]).

Additionally, a scalable load balancing case study is presented in [16], having a Mealy-type semantics. For comparison and usage with ACACIA and LILY, the examples have been transformed to Moore-type semantics by prefixing all occurrences of output atomic propositions with an LTL next-time operator.

4 Observations on the differences and similarities of the synthesis tools

We continue with a discussion of the similarities and differences between the synthesis tools. The arguments to follow form the foundation of the experimental evaluation frameworks proposed in Section 5.

4.1 The incomparable scopes & semantics of the tools

When comparing the tools considered in this paper, it is striking that all four tools have incompatible specification languages. Only ACACIA 2009 and LILY have the same input specification format (however, ACACIA 2010 adds local assumptions to the input language which cannot be interpreted correctly by LILY).

It is fair to raise the question why this is the case, given the fact that the number of benchmark sets for synthesis is rather low, so one would expect that the scopes and semantics are compatible in order to have as many benchmarks available as possible. In this paper, we conjecture that the reason for this situation is that *the scopes of the tools are strongly adapted to the techniques implemented*, but whenever the choice does not matter, as close to the literature as possible.

We begin our discussion of this observation with the tool ANZU. The generalised reactivity(1) synthesis technique is implementable in both Mealy and Moore semantics. The tool ANZU uses Mealy semantics, as the description of the synthesis algorithm in [57]. The assumptions and guarantees allowed are precisely those that can be processed by the algorithm without giving up the idea that the state space of the underlying game is the set of input and output variable valuations. This can be seen from the fact that from the formula types given in Section 3.1, type (1) is only an initial state condition, and type (2) can be encoded into the transition relation of a game with such a structure. Formulas of type (3) are precisely those that can then be given to the actual solving process as liveness parameters.

In contrast to ANZU, LILY uses Moore-type semantics and some PSL-like input file syntax. Since LILY bases on tree automaton techniques, this is not surprising: using a Mealy-type semantics in the context of tree automata would require that the labelling of the initial node of a computation tree (that is either accepted or rejected by the tree automaton) is ignored, as the node labels represent the output of the system. On a theoretical level, such a definition would look unnecessarily awkward, which is why the Moore-type semantics are usually preferred in this context.

As LILY, ACACIA uses a Moore-type semantics and the same syntax as LILY. According to [22], the authors wanted to keep ACACIA 2009 comparable to LILY. An additional reason for keeping the Moore-type semantics is the better applicability of the results: specifications that are found to be realisable in the Moore-type semantics are also realisable in the Mealy-type semantics, but not vice versa. Only in ACACIA 2010, the input language is extended in order to accommodate the new features proposed in [23].

UNBEAST, on the other hand, uses a Mealy-type semantics and has its own XML-based input file format. In [16], it has been argued that specifications often become shorter and thus the (edge-labelled) Büchi automata become smaller in the Mealy setting, which is beneficial for a BDD-based approach. For example, when specifying some immediate output consequences of some input such as $\mathsf{G}(r \rightarrow g)$ for some input set $\{r\}$ and output set $\{g\}$, taking the Moore semantics would require the introduction of a next-time operator into the formula, which would be reflected in the automaton size. The XML-based input language has been used in order to circumvent the necessity for a complicated formula parser, but also to make the operator precedences explicit.

4.2 Comparability of the examples

The arguments from the preceding subsection explain why the scopes and semantics of the tools are different. Nevertheless, it does not explain why different specification sets have been used, as benchmarks for ANZU could be converted to benchmarks for the other tools and the conversion between Mealy- and Moore-type semantics is rather simple. Still, the only case in which benchmarks were converted for an experimental evaluation was in [16], where LILY's and ACACIA's examples were used for evaluating UNBEAST (and vice versa). In some cases, benchmarks have been rewritten, e.g., the IBM generalised buffer specification, which was used to evaluate ANZU, has been altered in [23] to a form in which the assumptions were made local. Also, there are two other publications [66, 51] reporting on experimental results for synthesis approaches using generalized parity games. In both of them, the feasibility of their approaches is shown using different reformulations of the AMBA arbiter example that was also used for ANZU.

An explanation for this fact was given in [67, 66, 5]. In fact, the GR(1) synthesis approach implemented in ANZU can accommodate all types of assumptions and guarantees that are representable as deterministic Büchi automata (DBA) [5, 15]. In order to fit into the input language of ANZU, however, the output bit set of the system to be synthesized has to be extended by state bits of the automaton. Somenzi and Sohail coined the term "pre-synthesis" for such an encoding, as converting an assumption or guarantee to such an automaton and encoding it into some output bits in a good way is a problem on its own for BDD-based techniques (see, e.g., [31, 26]). Thus, a lot of effort has been put into a good reformulation of the problem description before checking realisability. An equivalent approach to using DBAs is to introduce so-called *auxiliary signals* (or auxiliary variables) into the design [30, 8, 7]. It has been noted that rewriting a specification using different signals can significantly speed up the synthesis process [30, 8] and for the AMBA AHB specification, this has also been done. As a consequence, it is not surprising that a specification for which pre-synthesis was performed and that has been optimized towards ANZU, neither the authors of ACACIA nor UNBEAST (nor the authors of the works using generalized parity automata [66, 51]) used the AMBA AHB arbiter specifications in the form provided with the ANZU tool for comparisons.

With respect to the fact that the IBM Generalised Buffer example has been altered for usage with ACACIA 2010, the situation is similar: in the original specification, the assumptions were not localised; defining the scope of the assumptions was simply not an issue in this case. As soon as techniques are introduced that can make use of such local assumptions, the situation changes.

4.3 Complexity of the workflows

Except for ANZU, the workflows, i.e., the numbers and orders of computation steps in the realisability checking process, of the tools discussed here are rather complicated. LILY and ACACIA 2009 first convert the specification to a universal Büchi automaton and then perform, for some successively increasing bound value, a realisability check over this automaton. In order to also detect unrealisable specifications, the check must additionally be ran for the negated specification with a conversion between the two semantics types in parallel. The workflow of UNBEAST is similar. In contrast to many other formal methods experimental evaluations, this whole process is relatively complicated and might easily appear less compelling than more simple schemes that are used in, for example, SAT solvers.

It is fair to conjecture that future workflows will even be more complicated. Take for example, ANZU, which has a relatively straight-forward workflow. As it has been argued that the generalised reactivity(1) synthesis approach that is used in this tool could handle all assumptions and guarantees that are repre-

sentable by deterministic Büchi automata, developing a preprocessor that takes LTL specifications of this kind and produces equivalent ANZU specifications appears to be worthwhile to write. However, such a preprocessor would have a very complicated workflow. After converting the assumptions and guarantees to Büchi automata, these have to be determinised (whenever possible), using an external tool like LTL2DSTAR [40]. Afterwards, it is possibly wise to try some exhaustive minimisation method for these automata [15]. Then, the automata also have to be symbolically encoded [31, 26]. Furthermore, the time spent on optimising the automata has to be balanced against the overall computation time in order to avoid running out of time in the automaton optimisation step.[2] All in all, these aspects make the whole synthesis process quite complicated and arguably, less compelling than other approaches, which ultimately reduces the publishability of any result on such a workflow, which in turn leads to little incentive to perform research or write tools in this area.

5 Providing a framework for future evaluations

The preceding sections discussed the difficulties of composing meaningful experimental evaluations of synthesis tools. Nevertheless, as benchmarking is often considered to be the only way to distinguish promising ideas from the ones that are likely not to be useful (see, e.g., [71]), in this section, we propose *three evaluation schemes* for each of the problems of *using appropriate benchmarks* and *dealing with complex workflows* whose compositions respect the difficulties discussed earlier. The schemes are ordered from the minimum requirement to show that a new technique is worthwhile considering to the "superior" scheme that demonstrates clear advantages over previous techniques.

5.1 Benchmarking

5.1.1 Comparison using the home field advantage

It is fair to say that a new approach should beat older approaches at least in the cases in which it has a natural advantage. This is typically shown by taking some example specification that falls into the class of systems the new approach is intended to be applied to, applying a prototype implementation of the approach to it, and showing that previous tools perform worse using an automatic, ingenuine, conversion to the semantics/scopes of the previous tools. This means in particular to convert between Mealy- and Moore-type semantics if applicable. Competitor tools which can only handle a subset of the language of the new prototype tool need not be considered.

5.1.2 Comparison from a neutral view-point

One problem of benchmarking tools with different scopes and semantics against the same examples is that *specifications are typically geared towards the usage with a certain tool*. A typical example is the pre-synthesis process discussed in Section 4.2 that ensures that the specification of a system falls into the class handled by the GR(1) synthesis tools. After this has been done, the specification is not only suitable but also optimised for such a tool. As many signalling bits are introduced in the process, tools like LILY and ACACIA 2009 that are explicit in the input and output bit valuations have problems with handling

[2] In benchmark comparisons, it is customary to restrict the running times of the tools. Such a time restriction is the typical answer to the problem that in most experimental evaluations, there are some benchmark/tool combinations that do not yield a result even after days or weeks of computation time.

such pre-synthesized specifications even in cases in which they can deal with the non-pre-synthesized versions.

A similar situation arises for example when localising the assumptions (as discussed in Section 4.2): doing so is beneficial for ACACIA 2010, but renders the optimisations of UNBEAST unusable as the input is then no longer in the $(\bigwedge \text{assumptions}) \rightarrow (\bigwedge \text{guarantees})$ form.

As a solution to this problem, we propose the following scheme: given a setting, the specification is written for all tools to be compared individually, taking care of their specialities. If the prototype tool of a new approach performs better in such a situation than previous tools, it is clear that the techniques proposed have their merits if used correctly when modelling a specification. It should be noted, however, that this scheme favours tools that require some form of pre-synthesis: by rewriting the specification for the simpler tool in a smart way, its performance can often greatly be increased. As an example, we refer to the work on rewriting the AMBA AHB bus arbiter specification [30].

5.1.3 Beating the other tools where they have a natural advantage

As a third scheme, we propose that if a prototype implementation of some approach can beat other tools on benchmark suites on which they have a natural advantage, this should suffice to show the merits of a new approach without doubt. In order to do so, one would typically use an automatic converter between the scope and semantics (if necessary) of the other tool and the scope and semantics of the new prototype tool to import benchmarks originally written for the other tool. The converter must not apply sophisticated optimisations on the specification. As an example, converting the LTL formula $\mathsf{GFX}p$ to $\mathsf{GF}p$ for some atomic proposition p during the adaptation of the Mealy/Moore-type semantics should be considered to be fair, whereas rewriting a guarantee into a simpler one that is only equivalent if the given assumptions also hold is probably too complex for this scheme.

5.2 Complex workflows

5.2.1 Basic scheme

In order to combat the problem of having workflows that involve multiple steps that can be be skipped without obstructing the steps following (like for example automaton optimisations), we propose the following scheme: for successively increasing timeout values (using a reasonable granularity), the synthesis approach is performed using the given timeout value for *all* individual sub-steps involved until the respective tool execution yields an answer. If the least timeout value that leads to a result for the new technique is lower than the least such timeout value for previous approaches, it is shown that the new approach has some merits.

5.2.2 Advanced scheme

As an extension to the basic scheme, it is worthwhile to show that the timeout value obtained in the basic scheme does not make the old approaches look bad unnecessarily. Let A be the least timeout value (for every step of the workflow) tried such that the prototype tool of the new approach terminates with an answer. Let B be the overall running time of the process. If it can be shown that the other approaches do not even terminate with a timeout of B for each step, additional justification for the new approach is obtained. Of course, many intermediate variations between the basic and advanced schemes are possible.

5.2.3 Simple scheme

Probably the most convincing way to solve the problem of having complex workflows is to set static timeouts for the individual steps of the workflow and to just measure the overall running time. If it is better than those of other tools, this clearly shows the efficiency of the new approach. Obviously, this scheme is hard to follow when comparing against an other approach that has itself many steps which introduce a need for individual timeouts if the author of the other tool has not provided good values for these. Additionally, special care must be taken not to "overfit" [21] the timeouts for the individual steps – tuning these values for the new prototype tool against a benchmark set and then evaluating on the same set against the other tools in a publication is not fair and can be considered to be scientifically unsound.

6 Conclusion

In this paper, we discussed the problems of experimentally evaluating a synthesis tool. We discussed three major issues: the incomparable semantics and scopes of the tools, the bad comparability of the tools with respect to the benchmarks available and the complexities of the workflows. Three evaluation schemes to combat the first two of these problems and three schemes to account for the complex workflows have been presented.

While the workflow evaluation schemes cannot fully remove the problem that experimental evaluations using these are often not fully compelling to the reader of a scientific publication, they introduce means of comparing tools if they have parts in their workflow that may time out without prohibiting later steps (like, e.g., automaton optimisation). We must admit that currently, there is no synthesis tool that performs such steps. However, this on its own is an interesting fact: due to the immense set of techniques proposed in the literature for reducing the sizes and numbers of automata representing the overall specification, operations such as determining whether a guarantee is actually necessary in a specification or more complicated automaton minimisation techniques are not used in current tools yet even though the theory behind these operations has been established [33, 15, 18, 12]. Thus, we hope that the three workflow evaluation schemes proposed help to level the way to further advances in this area.

As a final note, we would like to defend the argumentation in this paper against the point of view that establishing a common file format with its clearly defined semantics and scope and requiring all future tools to use it as a basis is a way to fight the benchmarking problem. We have shown in Section 4 that the choice of techniques affects the choice of the semantics of a tool. Thus, picking one particular scope and semantics would drive the evolution of the tools and thus also the theory into a certain direction while ignoring possibilities apart from techniques not suitable for the scope and semantics agreed upon. However, as even if leaving this consideration apart, the form of a "typical" specification in practice (conjunction of guarantees [23, 45, 51] vs. $(\bigwedge \text{assumptions}) \rightarrow (\bigwedge \text{guarantees})$ form [7, 16, 8, 5, 30]) is not agreed upon, it is fair to argue that consensus will not be reached within the next few years. Also, an implicit or explicit requirement that tools with a complex workflow should always be evaluated in a way similar to the simple scheme proposed here is highly problematic: due to the low number of meaningful benchmarks, fixing good values as timeouts for the intermediate steps without overfitting for the concrete set of benchmarks in the evaluation is hardly possible. As a result, such a requirement would basically rule out complex optimisations a-priori (or require cheating by an author in the paper by overfitting), which is highly questionable for practical approaches to a problem that is, after all, still 2EXPTIME-complete.

Acknowledgements

This work was supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). The author wants to thank Barbara Jobstmann and Emmanuel Filiot for helpful comments on the descriptions of the techniques employed in ANZU, LILY and ACACIA.

References

[1] Martín Abadi, Leslie Lamport & Pierre Wolper (1989): *Realizable and Unrealizable Specifications of Reactive Systems*. In Ausiello et al. [3], pp. 1–17, doi:10.1007/BFb0035748.

[2] ARM Ltd. (1999): *AMBA™ specification (rev. 2)*. Available at www.arm.com.

[3] Giorgio Ausiello, Mariangiola Dezani-Ciancaglini & Simona Ronchi Della Rocca, editors (1989): *Automata, Languages and Programming, 16th International Colloquium, (ICALP)*. *LNCS* 372, Springer.

[4] Armin Biere, Marijn Heule, Hans van Maaren & Toby Walsh, editors (2009): *Handbook of Satisfiability*. IOS Press.

[5] Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger & Barbara Jobstmann (2010): *Robustness in the Presence of Liveness*. In Touili et al. [72], pp. 410–424, doi:10.1007/978-3-642-14295-6_36.

[6] Roderick Bloem, Alessandro Cimatti, Karin Greimel, Georg Hofferek, Robert Könighofer, Marco Roveri, Viktor Schuppan & Richard Seeber (2010): *RATSY - A New Requirements Analysis Tool with Synthesis*. In Touili et al. [72], pp. 425–429, doi:10.1007/978-3-642-14295-6_37.

[7] Roderick Bloem, Stefan Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli & Martin Weiglhofer (2007): *Interactive presentation: Automatic hardware synthesis from specifications: a case study*. In Rudy Lauwereins & Jan Madsen, editors: *DATE*. ACM, pp. 1188–1193, doi:10.1145/1266366.1266622.

[8] Roderick Bloem, Stefan Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli & Martin Weiglhofer (2007): *Specify, Compile, Run: Hardware from PSL*. *Electr. Notes Theor. Comput. Sci.* 190(4), pp. 3–16, doi:10.1016/j.entcs.2007.09.004.

[9] Roderick Bloem, Barbara Jobstmann & Martin Weiglhofer (2007): *Anzu*. http://www.ist.tugraz.at/staff/jobstmann/anzu/.

[10] Randal E. Bryant (1986): *Graph-Based Algorithms for Boolean Function Manipulation*. *IEEE Trans. Computers* 35(8), pp. 677–691.

[11] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen & Didier Lime (2005): *Efficient On-the-Fly Algorithms for the Analysis of Timed Games*. In Martín Abadi & Luca de Alfaro, editors: *CONCUR*. *LNCS* 3653, Springer, pp. 66–80, doi:10.1007/11539452_9.

[12] Lorenzo Clemente & Richard Mayr (2010): *Multipebble Simulations for Alternating Automata - (Extended Abstract)*. In Paul Gastin & François Laroussinie, editors: *CONCUR*. *LNCS* 6269, Springer, pp. 297–312, doi:10.1007/978-3-642-15375-4_21.

[13] Laurent Doyen, Emmanuel Filiot, Naiyong Jin & Jean-François Raskin (2009): *Acacia - LTL Realizability Check and Winning Strategy Synthesis using Antichains*. http://www.antichains.be/acacia/.

[14] Alexandre Duret-Lutz & Denis Poitrenaud (2004): *SPOT: An Extensible Model Checking Library Using Transition-Based Generalized Büchi Automata*. In Doug DeGroot, Peter G. Harrison, Harry A. G. Wijshoff & Zary Segall, editors: *MASCOTS*. IEEE Computer Society, pp. 76–83.

[15] Rüdiger Ehlers (2010): *Minimising Deterministic Büchi Automata Precisely Using SAT Solving*. In Strichman & Szeider [70], pp. 326–332, doi:10.1007/978-3-642-14186-7_28.

[16] Rüdiger Ehlers (2010): *Symbolic Bounded Synthesis*. In Touili et al. [72], pp. 365–379, doi:10.1007/978-3-642-14295-6_33.

[17] Rüdiger Ehlers (2010): *Unbeast – Symbolic Bounded Synthesis*. http://react.cs.uni-saarland.de/tools/unbeast/.

[18] Rüdiger Ehlers & Bernd Finkbeiner (2010): *On the Virtue of Patience: Minimizing Büchi Automata*. In van de Pol & Weber [62], pp. 129–145, doi:10.1007/978-3-642-16164-3_10.

[19] Cindy Eisner & Dana Fisman (2006): *A Practical Introduction to PSL (Series on Integrated Circuits and Systems)*. Springer-Verlag.

[20] E. Allen Emerson & A. Prasad Sistla, editors (2000): *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*. *LNCS* 1855, Springer.

[21] Emanuel Falkenauer (1998): *On Method Overfitting*. *J. Heuristics* 4(3), pp. 281–287.

[22] Emmanuel Filiot, Naiyong Jin & Jean-François Raskin (2009): *An Antichain Algorithm for LTL Realizability*. In Ahmed Bouajjani & Oded Maler, editors: *CAV*. *LNCS* 5643, Springer, pp. 263–277, doi:10.1007/978-3-642-02658-4_22.

[23] Emmanuel Filiot, Naiyong Jin & Jean-François Raskin (2010): *Compositional Algorithms for LTL Synthesis*. In Ahmed Bouajjani & Wei-Ngan Chin, editors: *ATVA*. *LNCS* 6252, Springer, pp. 112–127, doi:10.1007/978-3-642-15643-4_10.

[24] Bernd Finkbeiner & Sven Schewe (2005): *Uniform Distributed Synthesis*. In: *LICS*. IEEE Computer Society, pp. 321–330, doi:10.1109/LICS.2005.53.

[25] Michael J. Fischer & Richard E. Ladner (1979): *Propositional Dynamic Logic of Regular Programs*. *J. Comput. Syst. Sci.* 18(2), pp. 194–211.

[26] Riccardo Forth & Paul Molitor (2000): *An efficient heuristic for state encoding minimizing the BDD representations of the transistion relations of finite state machines*. In: *ASP-DAC*. ACM, pp. 61–66.

[27] Paul Gastin & Denis Oddoux (2001): *Fast LTL to Büchi Automata Translation*. In Gérard Berry, Hubert Comon & Alain Finkel, editors: *CAV*. *LNCS* 2102, Springer, pp. 53–65.

[28] Allen Van Gelder & Daniel Le Berre, editors (2010): *Pragmatics of SAT*. Workshop at the Federated Logic Conference (FLoC) 2010, Edinburgh.

[29] Naghmeh Ghafari, Alan J. Hu & Zvonimir Rakamaric (2010): *Context-Bounded Translations for Concurrent Software: An Empirical Evaluation*. In van de Pol & Weber [62], pp. 227–244, doi:10.1007/978-3-642-16164-3_17.

[30] Yashdeep Godhal, Krishnendu Chatterjee & Thomas A. Henzinger (2010): *Synthesis of AMBA AHB from Formal Specification*. *CoRR* abs/1001.2811. Available at http://arxiv.org/abs/1001.2811.

[31] Wilsin Gosti, Tiziano Villa, Alexander Saldanha & Alberto L. Sangiovanni-Vincentelli (2007): *FSM Encoding for BDD Representations*. *Applied Mathematics and Computer Science* 17(1), pp. 113–124, doi:10.2478/v10006-007-0011-6.

[32] Erich Grädel, Wolfgang Thomas & Thomas Wilke, editors (2002): *Automata, Logics, and Infinite Games: A Guide to Current Research*. *LNCS* 2500, Springer.

[33] Karin Greimel, Roderick Bloem, Barbara Jobstmann & Moshe Y. Vardi (2008): *Open Implication*. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir & Igor Walukiewicz, editors: *ICALP (2)*. *LNCS* 5126, Springer, pp. 361–372, doi:10.1007/978-3-540-70583-3_30.

[34] Aidan Harding, Mark Ryan & Pierre-Yves Schobbens (2005): *A New Algorithm for Strategy Synthesis in LTL Games*. In Nicolas Halbwachs & Lenore D. Zuck, editors: *TACAS*. *LNCS* 3440, Springer, pp. 477–492.

[35] Thomas A. Henzinger & Nir Piterman (2006): *Solving Games Without Determinization*. In Zoltán Ésik, editor: *CSL*. *LNCS* 4207, Springer, pp. 395–410, doi:10.1007/11874683_26.

[36] IBM Research: *RuleBase formal verification tool tutorial*. https://www.research.ibm.com/haifa/projects/verification/RB_Homepage/.

[37] Barbara Jobstmann & Roderick Bloem (2006): *Lily - a LInear Logic sYnthesizer*. http://www.iaik.tugraz.at/content/research/design_verification/lily/.

[38] Barbara Jobstmann & Roderick Bloem (2006): *Optimizations for LTL Synthesis*. In: *FMCAD*. IEEE Computer Society, pp. 117–124, doi:10.1109/FMCAD.2006.22.

[39] Barbara Jobstmann, Stefan Galler, Martin Weiglhofer & Roderick Bloem (2007): *Anzu: A Tool for Property Synthesis*. In Werner Damm & Holger Hermanns, editors: *CAV*. *LNCS* 4590, Springer, pp. 258–262, doi:10.1007/978-3-540-73368-3_29.

[40] Joachim Klein & Christel Baier (2006): *Experiments with deterministic ω-automata for formulas of linear temporal logic*. *Theor. Comput. Sci.* 363(2), pp. 182–195, doi:10.1016/j.tcs.2006.07.022.

[41] Uri Klein & Amir Pnueli (2010): *Revisiting Synthesis of GR(1) Specifications*. In: *HVC*. *LNCS* 6504.

[42] Stephan Kottler (2010): *SAT Solving with Reference Points*. In Strichman & Szeider [70], pp. 143–157, doi:10.1007/978-3-642-14186-7_13.

[43] James H. Kukula & Thomas R. Shiple (2000): *Building Circuits from Relations*. In Emerson & Sistla [20], pp. 113–123.

[44] Orna Kupferman (2006): *Avoiding Determinization*. In: *LICS*. IEEE Computer Society, pp. 243–254, doi:10.1109/LICS.2006.15.

[45] Orna Kupferman, Nir Piterman & Moshe Y. Vardi (2006): *Safraless Compositional Synthesis*. In Thomas Ball & Robert B. Jones, editors: *CAV*. *LNCS* 4144, Springer, pp. 31–44, doi:10.1007/11817963_6.

[46] Orna Kupferman & Moshe Y. Vardi (1999): *Church's problem revisited*. *Bulletin of Symbolic Logic* 5(2), pp. 245–263. Available at http://www.math.ucla.edu/~asl/bsl/0502/0502-004.ps.

[47] Orna Kupferman & Moshe Y. Vardi (2001): *Model Checking of Safety Properties*. *Formal Methods in System Design* 19(3), pp. 291–314.

[48] Orna Kupferman & Moshe Y. Vardi (2001): *Synthesizing Distributed Systems*. In: *LICS*.

[49] Orna Kupferman & Moshe Y. Vardi (2005): *Safraless Decision Procedures*. In: *FOCS*. IEEE, pp. 531–542, doi:10.1109/SFCS.2005.66.

[50] Xinxin Liu & Scott A. Smolka (1998): *Simple Linear-Time Algorithms for Minimal Fixed Points (Extended Abstract)*. In Kim G. Larsen, Sven Skyum & Glynn Winskel, editors: *ICALP*. *LNCS* 1443, Springer, pp. 53–66, doi:10.1007/978-3-540-64781-2_53.

[51] A. Morgenstern (2010): *Symbolic Controller Synthesis for LTL Specifications*. Ph.D. thesis, Department of Computer Science, University of Kaiserslautern, Germany.

[52] A. Morgenstern & K. Schneider (2010): *Exploiting the Temporal Logic Hierarchy and the Non-Confluence Property for Efficient LTL Synthesis*. In A. Montanari, M. Napoli & M. Parente, editors: *GandALF*. *EPTCS* 25, pp. 89–102, doi:10.4204/EPTCS.25.11.

[53] Silvia M. Müller & Wolfgang J. Paul (2000): *Computer architecture: complexity and correctness*. Springer.

[54] Alexander Nadel & Vadim Ryvchin (2010): *Assignment Stack Shrinking*. In Strichman & Szeider [70], pp. 375–381, doi:10.1007/978-3-642-14186-7_35.

[55] Carsten Sinz (organiser): *SAT-Race 2010*. http://baldur.iti.uka.de/sat-race-2010/.

[56] Ian Parberry (1994): *A Guide for New Referees in Theoretical Computer Science*. *Inf. Comput.* 112(1), pp. 96–116.

[57] Nir Piterman, Amir Pnueli & Yaniv Sa'ar (2006): *Synthesis of Reactive(1) Designs*. In E. Allen Emerson & Kedar S. Namjoshi, editors: *VMCAI*. *LNCS* 3855, Springer, pp. 364–380, doi:10.1007/11609773_24.

[58] Amir Pnueli (1977): *The Temporal Logic of Programs*. In: *FOCS*. IEEE, pp. 46–57.

[59] Amir Pnueli & Roni Rosner (1989): *On the Synthesis of a Reactive Module*. In: *POPL*. pp. 179–190.

[60] Amir Pnueli & Roni Rosner (1989): *On the Synthesis of an Asynchronous Reactive Module*. In Ausiello et al. [3], pp. 652–671.

[61] Amir Pnueli, Yaniv Sa'ar & Lenore D. Zuck (2010): *Jtlv: A Framework for Developing Verification Algorithms*. In Touili et al. [72], pp. 171–174, doi:10.1007/978-3-642-14295-6_18.

[62] Jaco van de Pol & Michael Weber, editors (2010): *Model Checking Software - 17th International SPIN Workshop, Enschede, The Netherlands, September 27-29, 2010*. *LNCS* 6349, Springer, doi:10.1007/978-3-642-16164-3.

[63] Kristin Y. Rozier & Moshe Y. Vardi (2010): *LTL satisfiability checking*. *STTT* 12(2), pp. 123–137, doi:10.1007/s10009-010-0140-3.

[64] Shmuel Safra (1989): *Complexity of Automata on Infinite Objects*. Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel.

[65] Sven Schewe & Bernd Finkbeiner (2007): *Bounded Synthesis*. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino & Yoshio Okamura, editors: *ATVA*. *LNCS* 4762, Springer, pp. 474–488, doi:10.1007/978-3-540-75596-8_33.

[66] Saqib Sohail & Fabio Somenzi (2009): *Safety first: A two-stage algorithm for LTL games*. In: *FMCAD*. IEEE, pp. 77–84, doi:10.1109/FMCAD.2009.5351138.

[67] Saqib Sohail, Fabio Somenzi & Kavita Ravi (2008): *A Hybrid Algorithm for LTL Games*. In Francesco Logozzo, Doron Peled & Lenore D. Zuck, editors: *VMCAI*. *LNCS* 4905, Springer, pp. 309–323, doi:10.1007/978-3-540-78163-9_26.

[68] Fabio Somenzi (2009): *CUDD: CU Decision Diagram Package Release 2.4.2*.

[69] Fabio Somenzi & Roderick Bloem (2000): *Efficient Büchi Automata from LTL Formulae*. In Emerson & Sistla [20], pp. 248–263.

[70] Ofer Strichman & Stefan Szeider, editors (2010): *Theory and Applications of Satisfiability Testing (SAT)*. *LNCS* 6175, Springer, doi:10.1007/978-3-642-14186-7.

[71] Walter F. Tichy (1998): *Should Computer Scientists Experiment More? IEEE Computer* 31(5), pp. 32–40.

[72] Tayssir Touili, Byron Cook & Paul Jackson, editors (2010): *The 22nd International Conference on Computer Aided Verification (CAV)*. *LNCS* 6174, Springer, doi:10.1007/978-3-642-14295-6.

[73] Moshe Y. Vardi (1995): *An Automata-Theoretic Approach to Linear Temporal Logic*. In Faron Moller & Graham M. Birtwistle, editors: *Banff Higher Order Workshop*. *LNCS* 1043, Springer.

[74] Moshe Y. Vardi & Larry J. Stockmeyer (1985): *Improved Upper and Lower Bounds for Modal Logics of Programs: Preliminary Report*. In: *STOC*. ACM, pp. 240–251.