The Reactive Synthesis Competition: SYNTCOMP 2016 and Beyond

Swen Jacobs
Saarland University
Saarbrücken, Germany

Roderick Bloem
Graz University of Technology
Graz, Austria

We report on the design of the third reactive synthesis competition (SYNTCOMP 2016), including a major extension of the competition to specifications in full linear temporal logic. We give a brief overview of the synthesis problem as considered in SYNTCOMP, and present the rules of the competition as well as the current benchmark set. Furthermore, we give an outlook on further changes and extensions of the competition that are planned for the future.

1 Introduction

The automatic synthesis of reactive systems from formal specifications has been one of the major challenges of computer science for more than 50 years, and a number of fundamental approaches to solve the problem have been proposed [15,23,49,51]. For a long time, the impact of theoretical results on the practice of system design has been rather limited, due to the high worst-case complexity of synthesis from specifications in expressive temporal logics, and a lack of algorithms that solve the problem efficiently in the average case. Recently, there have been a number of new approaches that aim at more practical synthesis algorithms by either restricting the specification language [7,46], or by a smart exploration of the search space [26,29–31,33,53]. Moreover, there has been an increased interest in applications of reactive synthesis techniques, e.g., in robotics and cyber-physical systems, or for the synthesis of device drivers. [20,22,42,43,52] Despite this growing interest, there remains a divide between theoretical research and applications, due in some part to a missing infrastructure to compare synthesis tools, and a lack of incentive to build efficient and mature implementations (as noted by Ehlers [25]).

In 2014, the authors and Ehlers founded the reactive synthesis competition (SYNTCOMP) in order to foster the research in scalable and user-friendly implementations of synthesis techniques. The goals of SYNTCOMP are

i) to make synthesis tools comparable by establishing a *common benchmark format*,
ii) to facilitate the exchange of benchmarks in a *public benchmark repository*,
iii) to establish a dedicated platform for a *comprehensive and fair evaluation* of synthesis tools,
iv) to encourage the implementation of synthesis tools that can be used as *black-box solvers* in applications, and
v) to foster the *efficient implementation* of synthesis algorithms by regularly providing new and challenging benchmark problems, and comparing the performance of tools on these.

Since its inception, SYNTCOMP was held annually, and the first two iterations [35,36] were intentionally restricted to safety properties and a low-level specification format derived from the existing AIGER format [5,34], in order to have a low entry barrier for participants. We consider the competition to be a great success thus far: were before there were no two synthesis tools that used the same input language, there are now five tools from different research groups that entered the competition and can be compared

Submitted to:
SYNT 2016

in a fair and meaningful way, based on a common specification language. As a part of the SYNTCOMP effort, we maintain a public benchmark library[1] that now consists of several thousand benchmark instances from a wide range of domains, and is steadily growing. Moreover, SYNTCOMP has triggered an increased interest in the development of efficient synthesis tools and specification languages that relate to the competition, as witnessed by a growing number of publications on these topics [8, 11, 13, 14, 41] [2], including tools and research groups that have not participated in the competition thus far [21, 24, 47, 56].

SYNTCOMP 2016 introduces a major extension of the competition by dropping the restriction to low-level safety properties. To this end, we add separate competition tracks for the evaluation of synthesis tools on specifications in a high-level input format for full linear temporal logic (LTL), as well as for the popular GR(1) fragment of LTL. The specification format used for the new tracks is the *temporal logic synthesis format* (TLSF), recently introduced by Jacobs and Klein [37].

In this paper, we describe the design of SYNTCOMP 2016, with a focus on the extension to specifications in TLSF, and report on our plans for further extensions of the competition in the coming years. We describe the synthesis problem as considered in SYNTCOMP in Section 2, followed by a presentation of the design and rules of SYNTCOMP 2016 in Section 3. Although benchmark and tool submission was still open at the time of this writing, we give a preliminary report on benchmarks and participants in SYNTCOMP 2016 in Sections 4 and 5, respectively. Finally, in Section 6 we give our thoughts on possible and probable further extensions of SYNTCOMP in the future, as a basis for discussion.

2 Reactive Synthesis: A Brief Overview

We briefly summarize the reactive synthesis problem as it is considered in SYNTCOMP, including approaches that have been developed to solve it.

The Synthesis Problem. We consider the synthesis problem for reactive systems that can be represented as *finite-state machines*. The specifications we consider come in two forms: either as temporal logic formulas, more specifically in *linear- time temporal logic* (LTL) [48], over the sets of inputs and outputs of the system, or as an AIGER circuit [5, 34] with a single output, with a set of controllable and a set of uncontrollable inputs.

For specifications in LTL, the *realizability problem* is to decide whether there exists a finite-state machine that reads the inputs and produces outputs such that the specification is satisfied in all possible executions. For AIGER circuits, the *realizability problem* is to decide whether there exists a controller circuit that reads the controllable inputs and the current state of the specification circuit, and produces the controllable inputs of the specification circuit such that the single output of the circuit is never raised.

Given a realizable specification, the *synthesis problem* is to find an implementation that satisfies the specification. The synthesis problems we consider are equivalent to finding a winning strategy in infinite two-player games whose structure and winning strategies are determined by the specification [57]. For both kinds of specifications, solutions can be encoded into an AIGER circuit.

Important Fragments. There are several important fragments of LTL, differing in expressivity and in the complexity of the realizability and synthesis problems. For full LTL, these problems are 2EXPTIME-complete in the size of the specification formula. If we restrict specifications to safety properties, then

[1] Synthesis Competition Repository: `https://bitbucket.org/swenjacobs/syntcomp/`

[2] Moreover, the ideas from [59] were also used in the version of Simple BDD Solver that competed in SYNTCOMP 2015.

the problems are in PSPACE. Another important fragment that we consider in SYNTCOMP is GR(1) [7], which allows some restricted liveness properties in addition to simple safety properties. For GR(1), the synthesis problem is in EXPTIME.[3]

Synthesis Algorithms. There are a number of existing algorithms to solve the synthesis problem, based on two fundamental approaches. The first approach, by Büchi and Landweber [15], works by translation into a deterministic Büchi game, and solving it. The second approach, by Rabin [51], works by translation into a tree automaton, and solving its emptyness problem.

In recent years, many algorithms for solving synthesis problems more efficiently have been proposed. We mention a few prominent approaches. Bounded synthesis [33] searches incrementally for solutions up to a certain size. An algorithm based on bounding liveness properties and a symbolic representation by antichains has been implemented in the synthesis tool Acacia [27,28]. Other algorithms try to exploit the structure of commonly occuring specifications, and propose incremental or compositional ways to solve the problem [29,40,53].

For safety properties, efficient algorithms can be implemented using BDDs and a fixpoint construction over the uncontrollable predecessors of the unsafe states. For GR(1), there is a similar algorithm, using a nested fixpoint construction [7]. A more detailed introduction into approaches to solve safety games can be found in the report of SYNTCOMP 2014 [35].

3 SYNTCOMP 2016: Rules and Setup

The basic idea of SYNTCOMP is that submitted tools are evaluated on a previously unknown set of benchmarks, without user intervention. Tools are then ranked with respect to number of problem instances that can correctly be solved within a given timeout. The competition is separated into tracks that correspond to the fragments of LTL mentioned in Section 2.

In the following, we first give an overview of the rules that are common to all tracks, and then go into some details for the separate tracks, with a focus on the changes made this year.

Tracks. The competition is divided into three main *tracks*, distinguished by the specification format: i) safety specifications in AIGER format, ii) full LTL specifications in TLSF, and iii) GR(1) specifications in TLSF. In all tracks, realizability is defined with respect to Mealy semantics, i.e., the outputs of an implementation can depend on the inputs without any delay.

In each track there are four different *subtracks*, distinguished by the task (realizability checking or synthesis) and by the execution mode (sequential or parallel). While in *realizability checking* the tools only need to return one bit of information, in *synthesis* they need to return a provably correct solution. In *sequential* categories, tools can only use one core of the CPU, and in *parallel* categories they can use multiple cores in parallel. Thus, we have twelve competitive subtracks overall.

Entrants. As in previous years, we ask participants to hand in their tools as source code, licensed for research purposes, accompanied by installation instructions and a short description of the system and the synthesis approach and optimizations it implements.

[3]More precisely, for GR(1) the size of the game arena is exponential in the size of the formula, and GR(1) games are solved in quadratic time in the size of the arena.

Each author can submit up to three different *tool configurations* per subtrack. Our experience from previous iterations suggests that this limit is a good compromise that allows some flexibility for the tool creators, while avoiding the flooding of the competition with too many configurations of the same tool.

The organizers commit to making reasonable efforts to install each tool, but reserve the right to reject entrants where installation problems cannot be resolved. This was not the case for any of previous iterations of the competition. In case of crashes or obviously wrong results we will allow submission of bugfixes, if time permits.

We encourage participants to visit the SYNT workshop and the CAV conference for the presentation of the SYNTCOMP results, but this is not a requirement for participation. The organizers reserve the right to submit their own tools, and did so in previous years.

Timeout. In sequential execution mode, the timeout for each problem is 3600s of CPU time. In the parallel mode, the timeout is 3600s of Wall Time.

Ranking. Competition entrants will be ranked with respect to the number of problems that can be answered with a correct solution within the given timeout. Timeouts are not counted, and wrong results are punished by subtracting 4 points. Since most or all of the benchmarks will be available publicly before the competition, we do not expect such a punishment to be necessary.

Correctness in realizability subtracks is determined either by existing information about the realizability of the benchmark (possibly stored in the `STATUS` field of the specification [36]), or by a majority vote of all participating solvers if such information is not available. In the latter case, the execution platform for the experiments generates a notification that a previously unsolved problem has been solved, and the organizers inspect the problem to avoid errors. Correctness in the synthesis subtracks is determined by verification of the produced solution within a separate time limit of 3600s (for details see below).

Quality Metrics. The goal of synthesis is to obtain implementations that are not only correct, but also efficient. Therefore, in previous iterations of SYNTCOMP we also considered additional *quality rankings*, where correct solutions are additionally weighted based on their size. Since the rankings used in previous years gave unsatisfactory results, we will not have an official quality ranking this year. However, we will still analyze solutions with respect to their size and present our findings at the presentation of results and in the final report.

We plan to bring quality rankings back in future iterations of the competition, based on our experience from SYNTCOMP 2016 and our thoughts presented in Section 6.3.

Competition Setup. Like in previous years, SYNTCOMP 2016 is organized on the EDACC platform [3]. The competition runs on a set of machines at Saarland University, each with a single Intel XEON processor (E3-1271 v3, quad-core, 3.6GHz) and 32 GB RAM (PC1600, ECC), running a GNU/Linux system. Each node has a local 480GB SSD that can store temporary files. To ensure a high comparability and reproducability of our results, a complete machine will be reserved for each job, i.e., one synthesis tool (configuration) running one benchmark. Since all nodes are identical and no other tasks will run in parallel, no other limits than the timeout will be set.

Benchmark Selection. A subset of all available benchmarks will be selected for the competition. Like in the previous year [36], benchmarks will be separated into categories, and we will ensure that the different categories have approximately equal weight in the competition, and that the competition benchmarks represent a good distribution across different difficulties for each category.

3.1 Specific Rules for New Tracks

Specifications. In the LTL and GR(1) tracks, specifications are given in basic TLSF format. For tools that do not support TLSF directly, the organizers supply a number of translators to different existing formats in the SyFCo tool [1] (which will be installed on the competition machines). In the LTL track, specifications are interpreted according to standard LTL semantics. In the GR(1) track, specifications are restricted to the GR(1) fragment and are to be interpreted under strict implication semantics. Additionally, we offer a translation to LTL with standard semantics, which allows LTL synthesis tools to compete in this track.

Output and Correctness. In the synthesis subtracks, tools must produce solutions in AIGER format if the specification is realizable. As a syntactical restriction, the sets of inputs and outputs of the TLSF file must be identical to the sets of inputs and outputs of the AIGER solution. Additionally, solutions will be model checked with existing LTL model checking tools.

Legacy Tools. For comparison, we plan to run legacy synthesis tools, non-competitive, in the LTL/GR(1) tracks. To this end, we will convert the TLSF specification to the native input format of the legacy tools, and use a wrapper script to make inputs and outputs conform to the standard format. Since this could be a significant amount of work for the synthesis subtracks, we will only consider the realizability problem for legacy tools. Legacy tools may include the latest available version of Unbeast and Lily in the LTL track, and Anzu in the GR(1) track (unless a new version of them is entered into the competition).

3.2 Specific Rules for AIGER safety track

Specifications. Specifications are given in the Extended AIGER Format for Synthesis [34,35], modeling a single safety property.

Output and Correctness. In the synthesis category, tools must produce solutions in AIGER format. These will be model checked with the best-performing model checkers from the single-safety track of the Hardware Model Checking Competition.

Since model checking turned out to be a significant challenge for some problem instances in previous years, we introduce another extension in SYNTCOMP 2016. As an alternative to full model checking, tools can output, in addition to their solution, a *winning region* of the system as a witness for correctness. If a winning region is returned, then correctness of the solution is determined by checking that the winning region is an inductive invariant for the given solution.

Legacy Tools. For comparison, we will run some of the entrants of SYNTCOMP 2014 and SYNTCOMP 2015 in the AIGER safety track. This allows us to highlight the progress of tools over the course of the last two years.

4 SYNTCOMP 2016: Benchmarks

At the time of writing, the collection of benchmarks for SYNTCOMP 2016 is not finished. We summarize the state of the SYNTCOMP benchmark library, and briefly describe the new benchmarks that have been added for 2016 thus far:

4.1 Existing Library

From previous years, we inherit a benchmark library in AIGER format with 20 different classes of benchmarks, consisting of 3105 different benchmark instances. The benchmarks come from diverse academic sources.

Initial Benchmark Set. Benchmarks added in the first year are described in more detail in the report of SYNTCOMP 2014 [35]. They consist of:

- ARMs AMBA specification and Intels Generalized Buffer specification (both described in previous work by Bloem et al. [7]),
- simple reactive systems such as traffic lights and arbiters of different complexity (taken from the test suite of synthesis tool LILY [39,40]),
- encodings of the `ltl2dba` and `ltl2dpa` problems that use the synthesis tool to obtain a deterministic Büchi automaton (dba) or deterministic parity automaton (dpa) from an LTL specification (taken from the Acacia+ benchmark set [12,27]),
- a load balancer benchmark (as originally described by Ehlers [25]),
- a "factory assembly line" benchmark that models two robot arms on an assembly line,
- a "moving robots" benchmark that models a robot that should avoid obstacles and reach a goal in two-dimensional space, and
- a number of basic building blocks of circuits, such as adders, bitshifters, counters, and multipliers, of different bit-width.

Benchmarks added in 2015. Benchmarks added in the second year are described in detail in the report of SYNTCOMP 2015 [36]. They consist of additional instances of some of the classes above, as well as:

- a benchmark class that models the scheduling of washing cycles under different constraints,
- benchmarks for the synthesis of drivers for a hard disk controller (based on a driver synthesis benchmark for the Termite synthesis tool [2,52]),
- the synthesis of a Huffman encoder (as described by Khalimov [41]),
- artificial benchmarks based on problems from the hardware model checking competition HWMCC [6, 16]; the original instances are unsafe and we ask the synthesis tool for a controller of a subset of the inputs such that the resulting system is safe,
- a set of benchmarks derived from HyperLTL model checking problems [32], where the synthesis tool is used to generate a witness for the given HyperLTL property,
- a set of benchmarks that uses the synthesis tool to generate matrix multiplication circuits of different bit-width, and possibly with repeated multiplication.

4.2 New Benchmarks

TLSF Benchmarks. For SYNTCOMP 2016, we have added a number of benchmarks in TLSF that have been present in AIGER format before. This includes the benchmarks that come with LTL synthesis tool LILY [39, 40], as well as those that come with Acacia+ [12, 27] (i.e., load balancer, ltl2dba/ltl2dpa, and a version of the generalized buffer), as mentioned above.

One of the great benefits of the TLSF format is that it supports parameterized benchmarks. The following pre-existing benchmarks have been added in a parameterized form:

- AMBA bus controller (parameterized in the number of masters that want to access the bus),
- generalized buffer (parameterized in the number of receivers), and
- load balancer (parameterized in the number of servers).

In addition, we have a number of new parametric benchmarks in TLSF:

- a number of arbiters of different complexity (parameterized in the number of choices),
- a set of new `ltl2dba` benchmarks, taken from recent work on the translation of LTL into automata [58] (these formulas are given in parametric form),
- a corresponding set of new `ltl2dpa` benchmarks (additionally parameterized in the number of colors of the parity automaton),
- a simple detector that checks whether all inputs are true eventually (parameterized in the number of inputs), and
- a compositional version of the AMBA benchmark that consists of a number of independent synthesis problems (some of them parameterized in the number of masters).

AIGER Benchmarks. For the AIGER track, we have added two sets of new benchmarks to the library this year:

- the new `ltl2dba` and `ltl2dpa` benchmarks mentioned above have been translated to AIGER format, using the LTL2AIG toolchain described in [35], for a fixed range of parameters and approximations of liveness by safety properties, and
- a set of additional derived benchmarks based on safety model checking in HWMCC, more specifically the 2012 edition of HWMCC.

5 SYNTCOMP 2016: Participants

At the time of writing, the deadline for participation is still one month away. From personal communication with previous SYNTCOMP participants and additional interested parties, we know that at least seven different research groups are working on tools for SYNTCOMP 2016.

Like last year, we will also run tools that entered previous competitions on the new benchmark collection, to get a better idea of the progress that is being made. For the TLSF-based tracks, we plan to run legacy tools (based on a translation from TLSF to their respective input formats) in addition to the participants.

6 The Future: Ideas for SYNTCOMP 2017 and Beyond

6.1 Compositional Specifications and Systems

Possible extensions of the specification format are in part also discussed in the format description [37]. Here, we focus on extensions of the competition, and what this means for the synthesis problems that need to be solved.

Compositional Specifications: Systems that need to be synthesized often consist of multiple components that can either be synthesized separately if specifications are completely local, or need to be synthesized such that the composed system additionally satisfies a global specification. The latter case is interesting for SYNTCOMP, and is currently not supported by the specification format.

Partial Implementations: When considering composed systems, a natural case of the synthesis problem is the synthesis of components for a system that is already partially implemented, i.e., where some components have a fixed implementation.

In some sense, this problem is already considered in the AIGER tracks of SYNTCOMP, as an AIGER file can contain both an implementation of a component, and a monitor automaton that raises an error output if the safety specification is violated.

In a component-based system, an existing implementation of a component could for example be given as an AIGER circuit. An extension of TLSF with such components could also be easily modified to generalize both the existing TLSF format and the existing AIGER format, as explained in the following.

Integration of both formats into one: If the supported format includes both compositional TLSF specifications and partial implementations as AIGER circuits, then the resulting format generalizes both the existing TLSF format (obviously), and the existing AIGER format: a given specification in AIGER format can simply be added as a component with a fixed implementation and a single output Error, where controllable inputs are assigned as outputs to the system to be synthesized, and the specification of the system is simply

$$\mathsf{G}\,\neg\mathsf{Error}.$$

Imperfect Information: Finally, compositional specifications lead to the synthesis problem under partial information, i.e., the components need to decide on their behavior without knowing all inputs or the full internal state of the other components. As Pnueli and Rosner have shown [50], the synthesis problem is undecidable under partial information, already for pure safety specifications. However, there have been a number of approaches to solve instances of the problem [10, 31, 33, 44, 45], and it would be interesting to include it into the competition at some point.

6.2 Synthesis Challenges

In its third year, SYNTCOMP is still in the process of natural growth, and is only establishing itself as a regular institution in the synthesis community. In some related research fields, competitions have been around for a long time, and there have been some unintentional adverse effects on the development of tools. On the one hand, a competition gives additional incentive for the development of efficient push-button tools, and positive effects of competitions on the quality and efficiency of tools have been observed [4, 16, 38, 55]. On the other hand, the specific design and rules of a competition may also discourage research on certain aspects of a problem, if they are *not* part of the competition. A long-running competition may also produce a number of very efficient and mature tools that discourage newcomers from entering the field.

Thus, as organizers of SYNTCOMP we have to admit to a responsibility for the research directions that we encourage or discourage by the design and the effects of the competition. One way to deal with the problems mentioned above would be flexible *synthesis challenges* that change from year to year (or every few years), and might be decided on by the community. Some of the tasks mentioned above, such as specialized quality metrics, could be offered as challenges for a limited time.

Another option is to provide potential participants with baseline solvers that already integrate the commonly accepted optimizations, such that the participants can focus on additional smart solutions, and don't have to implement all the basic features themselves. This approach could even be enforced in a special track, where participants must start from this common baseline, and are only allowed to make limited changes to the implementation that is supplied. An example of such an approach are the "Hack Tracks" of the SAT competition, where participants start from a given SAT solver and the difference between the baseline and their own implementation is limited to 1000 (non-space) characters.

6.3 Quality Ranking/Quantitative Aspects

As mentioned before, in synthesis we usually not only care about correctness of our implementations, but also about quantitative properties of the synthesized artifact, like its size, its reaction time to certain events, or possibly other aspects like energy efficiency.

Experience in Previous Competitions. In SYNTCOMP 2014 and 2015, we used different quality rankings based on the number of AND-gates in the solution, by comparing either against the size of other solutions in the given competition, or against the size of a reference solution. A comparison against a value that is not fixed before the competition means that the results (including the relative ranking of tools) may change when we add a tool. This is already undesirable in general, but in particular if we want to use the results of the competition to evaluate a tool that did not participate. Therefore, using a reference solution is in general preferable. However, reference solutions are not always available, since one of our goals is to have *new* and *challenging* benchmarks in every iteration of the competition. Therefore, in SYNTCOMP 2015 we adopted a mixed solution, which uses the size of a reference solution if available, and the size of the smallest solution in the current run otherwise. Evaluation of non-participating tools then requires to take this distinction into account.

However, even with this mixed approach, the results of the quality ranking were somewhat unsatisfactory, since the size of solutions effectively only played a small role, and was dominated by the number of problems that could be solved. This was due to two main reasons. First, points for the size of solutions were given according to a log_{10}-scale, i.e., for solution A to get one additional point compared to solution B, A had to be 10 times smaller than B. Probably a log_2-scale would be better if we want to emphasize the need for small implementations. Second, we compared the size of full solutions, which in the AIGER format includes the specification circuit. Since we have many problems with a very large specification, the size of the solution is often dominated by the size of the specification, and the size of the synthesized code does not make much of a difference. An option to repair this would be to compare the size of the synthesized code instead of the size of the full solution. In SYNTCOMP 2016, we will not have an official quality ranking, but we will experiment with this evaluation scheme to see if it should be used in the future.

Quality of Solutions for LTL specifications. With our extension of SYNTCOMP to specifications in LTL, one question is whether the same ranking scheme is also suitable and fair for the new tracks. In the AIGER-based track, input and output are both symbolically encoded, and we can reasonably expect tools to optimize with respect to this encoding. In the LTL-based tracks, the input is not symbolically encoded, and the output encoding will in many cases be an additional step to conform to the competition format. Therefore, the answer to the question of fairness and suitability is not obvious. On the other hand, one can argue that almost all solutions that are efficient by some different measure can also be encoded into a small symbolic AIGER representation. In SYNTCOMP 2016 we will experiment with the existing quality measures also for the TLSF-based tracks, and want to discuss the issue with the community at the SYNT workshop.

Quality Measures Beyond (Circuit) Size. There are many other natural quality measures for reactive systems. These include:

- the size of the reachable state space (either of the synthesized strategy, or of the solution that includes the specification circuit),
- the reaction time to certain actions/inputs of the environment, and
- more complex measures that assign a cost to certain actions of the system, e.g. a measure for *energy-efficiency*.

Specialized synthesis approaches that optimize a solution with respect to these measures exist [9, 17–19, 33, 60]. We want to discuss at the SYNT workshop whether any of them should be a standard quality measure in future competitions, or whether we should use optimization towards them as special challenges for some competitions.

6.4 Witnesses for Correctness

The problems we consider in SYNTCOMP are realizability of a specification and synthesis of a solution. While the production of solutions is optional in some other subfields of automated reasoning and computer-aided verification, it is at the heart of SYNTCOMP. Because of this, solutions themselves are natural witnesses for the correctness of a "realizable" statement. To verify that a solution is correct, it can be model checked against the specification.

However, there are a number of problems with this approach:

1. solutions of the synthesis problem can only be used as witnesses for correctness if the specification is realizable. If it is unrealizable, SYNTCOMP thus far did not require any witness of correctness.
2. model checking may not be easy for complex solutions and specifications. Even for pure safety specifications, we had a number of solutions in SYNTCOMP 2014 that could not be model checked, and an increased number in 2015.

In the following, we present some ideas how to handle these problems. Note that the proposed solutions are orthogonal and could be combined.

Witnesses for Unrealizability (Counter-strategies): To solve problem 1, the competition could include (either by default, or as separate track, or as a challenge) the computation of counter-strategies for unrealizable specifications.

The easiest way to investigate the performance of tools on this task would be to run the tool on the negated the specification and require a Moore-type implementation (instead of the usual Mealy-type).

However, combining the two tasks gives even more meaningful results, as in general we will not know whether our specification is realizable or not, and we want a witness of the fact regardless of the outcome.

Comprehensive Witnesses for Effective Correctness Proofs: Based on our findings in SYNTCOMP 2014 and 2015, we already introduced this year the possibility that tools (in the AIGER-based safety track) can give additional witness information that will make it easier to check correctness of the provided solution. For safety specifications, this information can simply be an inductive invariant of the produced solution, i.e., a set of states that does not contain error states and is such that the produced solution will never leave the set. The winning region of the system (computed by the standard fixpoint-based algorithm for safety games) is an example of an inductive invariant. In SYNTCOMP 2016, we allow that such an invariant is provided by the tool (also in the AIGER format), and we expect that this will solve the problems of verifying more complex solutions.

For specifications in LTL or GR(1), we expect the problem of verifiability to be even worse. Furthermore, comprehensive witnesses also need to contain more information if specifications are not restricted to safety, but may also contain liveness properties. Since the hardest part of the verification of liveness properties is essentially the construction of (some form of) suitable ranking function, it would be good if this ranking function could already be supplied by the synthesis tool. In case of GR(1), such ranking functions could probably have a rather simple form, which might boil down to a fixed unrolling of liveness properties and then effectively checking a safety property.

6.5 Technical Setup

Both SYNTCOMP 2015 and 2016 were run at Saarland University, on a small set of machines that were acquired specifically for this purpose. The benefit of this approach is that we were able to tailor the computers to the needs of our competition, which is CPU- and memory-intensive, but does not have a focus on parallelization. For instance, since none of the tools in the competition used more than 3 or 4 cores (in SYNTCOMP 2014 and 2015, respectively), we had a huge benefit from moving from machines with 16 CPU cores, but low sequential speed (in 2014), to machines with only 4 CPU cores, but nearly twice the sequential speed (in 2015 and 2016). Moreover, the organizers have full control over these machines (as opposed to machines that are operated and serviced by a third party), which makes the execution of the competition easier and more predictable.

However, the reduced number of available competition servers was already an issue last year. To cope with the problem, we reduced the number of benchmark instances that were tested in the competition overall.[4] Presumably, the capacity of the competition servers will be at its limit this year, and we probably have to reduce the number of benchmark instances per track to adjust for the additional tracks that are introduced this year. A bigger computing capacity is therefore desirable and possibly necessary.

This could be achieved in different ways, each with their own benefits and downsides. Increasing the number of machines in the current setup requires dedicated funding and local infrastructure, which may be hard to justify for a service that only runs 2-3 months per year. The other option is to use third-party machines, for example those provided by the StarExec platform [54]. The benefit would be essentially unlimited compute capacity, while the downside would be that we give up complete control over the machines and the execution of the competition, and have to adjust our technical setup to the infrastructure of that service (to a degree that is currently unknown to us).

[4]In fact, we significantly reduced the number from 569 to 250 in the realizability track, while increasing the number from 157 to 239 instances in the synthesis track.

7 Conclusions

The Reactive Synthesis Competition SYNTCOMP has been held annually since 2014. SYNTCOMP 2016 presents the biggest extension of the competition thus far, introducing additional tracks based on a high-level temporal logic specification format. SYNTCOMP is designed as a long-term effort that should be guided by feedback from the reactive synthesis community. In addition to the design of SYNTCOMP 2016, we have presented an overview of possible changes and extensions of the competition that could be considered in the future, and are looking forward to discuss the future of SYNTCOMP with the participants of SYNT 2016.

Acknowledgements. We thank Rüdiger Ehlers, Ayrat Khalimov, Felix Klein, Andrey Kupriyanov, Kim Larsen, Nir Piterman, Markus Rabe, and Leander Tentrup for interesting suggestions for the future of SYNTCOMP (and apologize if we forgot someone).

References

[1] *Synthesis Format Conversion Tool*. Available at `https://github.com/reactive-systems/syfco`.

[2] *Termite Driver Synthesis Tool*. Available at `http://ssrg.nicta.com.au/projects/TS/drivers/synthesis/`.

[3] Adrian Balint, Daniel Diepold, Daniel Gall, Simon Gerber, Gregor Kapler & Robert Retz (2011): *EDACC - An Advanced Platform for the Experiment Design, Administration and Analysis of Empirical Algorithms*. In: *LION 5. Selected Papers*, *LNCS* 6683, Springer, pp. 586–599, doi:10.1007/978-3-642-25566-3_46.

[4] Clark Barrett, Morgan Deters, Leonardo Mendonça de Moura, Albert Oliveras & Aaron Stump (2013): *6 Years of SMT-COMP*. *J. Autom. Reasoning* 50(3), pp. 243–277, doi:10.1007/s10817-012-9246-5.

[5] Armin Biere: *AIGER Format and Toolbox*. Available at `http://fmv.jku.at/aiger/`.

[6] Armin Biere: *Hardware Model Checking Competition*. `http://fmv.jku.at/hwmcc/`.

[7] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli & Y. Sa'ar (2012): *Synthesis of Reactive(1) designs*. *J. Comput. Syst. Sci.* 78(3), pp. 911–938, doi:10.1016/j.jcss.2011.08.007.

[8] R. Bloem, R. Könighofer & M. Seidl (2014): *SAT-Based Synthesis Methods for Safety Specs*. In: *VMCAI*, *LNCS* 8318, Springer, pp. 1–20, doi:10.1007/978-3-642-54013-4_1.

[9] Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger & Barbara Jobstmann (2009): *Better Quality in Synthesis through Quantitative Objectives*. In: *CAV*, *LNCS* 5643, Springer, pp. 140–156, doi:10.1007/978-3-642-02658-4_14.

[10] Roderick Bloem, Krishnendu Chatterjee, Swen Jacobs & Robert Könighofer (2015): *Assume-Guarantee Synthesis for Concurrent Reactive Programs with Partial Information*. In: *TACAS*, *LNCS* 9035, Springer, pp. 517–532, doi:10.1007/978-3-662-46681-0_50.

[11] Roderick Bloem, Uwe Egly, Patrick Klampfl, Robert Könighofer, Florian Lonsing & Martina Seidl (2016): *Satisfiability-Based Methods for Reactive Synthesis from Safety Specifications*. *CoRR* abs/1604.06204.

[12] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin & Jean-François Raskin (2012): *Acacia+, a Tool for LTL Synthesis*. In: *CAV*, *LNCS* 7358, Springer, pp. 652–657, doi:10.1007/978-3-642-31424-7_45.

[13] Romain Brenguier, Guillermo A. Pérez, Jean-François Raskin & Ocan Sankur (2014): *AbsSynthe: abstract synthesis from succinct safety specifications*. In: *SYNT*, *EPTCS* 157, pp. 100–116, doi:10.4204/EPTCS.157.11.

[14] Romain Brenguier, Guillermo A. Pérez, Jean-François Raskin & Ocan Sankur (2016): *Compositional Algorithms for Succinct Safety Games*. In: *SYNT*, *EPTCS* 202, pp. 98–111, doi:10.4204/EPTCS.202.7.

[15] J.R. Büchi & L.H. Landweber (1969): *Solving sequential conditions by finite-state strategies*. *Trans. Amer. Math. Soc.* 138, pp. 295–311, doi:10.2307/1994916.

[16] Gianpiero Cabodi, Carmelo Loiacono, Marco Palena, Paolo Pasini, Denis Patti, Stefano Quer, Danilo Vendraminetto, Armin Biere, Keijo Heljanko & Jason Baumgartner (2016): *Hardware Model Checking Competition 2014: An Analysis and Comparison of Solvers and Benchmarks*. Journal on Satisfiability, Boolean Modeling and Computation 9, pp. 135–172.

[17] Pavol Cerný, Krishnendu Chatterjee, Thomas A. Henzinger, Arjun Radhakrishna & Rohit Singh (2011): *Quantitative Synthesis for Concurrent Programs*. In: CAV, Lecture Notes in Computer Science 6806, Springer, pp. 243–259, doi:10.1007/978-3-642-22110-1_20.

[18] Pavol Cerný & Thomas A. Henzinger (2011): *From boolean to quantitative synthesis*. In: EMSOFT, ACM, pp. 149–154, doi:10.1145/2038642.2038666.

[19] Krishnendu Chatterjee, Thomas A. Henzinger, Barbara Jobstmann & Rohit Singh (2011): *QUASY: Quantitative Synthesis Tool*. In: TACAS, Lecture Notes in Computer Science 6605, Springer, pp. 267–271, doi:10.1007/978-3-642-19835-9_24.

[20] Yushan Chen, Xu Chu Ding, Alin Stefanescu & Calin Belta (2012): *Formal Approach to the Deployment of Distributed Robotic Teams*. IEEE Transactions on Robotics 28(1), pp. 158–171, doi:10.1109/TRO.2011.2163434.

[21] Ting-Wei Chiang & Jie-Hong R. Jiang (2015): *Property-Directed Synthesis of Reactive Systems from Safety Specifications*. In: ICCAD, IEEE, pp. 794–801, doi:10.1109/ICCAD.2015.7372652.

[22] Sandeep Chinchali, Scott C. Livingston, Ufuk Topcu, Joel W. Burdick & Richard M. Murray (2012): *Towards formal synthesis of reactive controllers for dexterous robotic manipulation*. In: ICRA, IEEE, pp. 5183–5189, doi:10.1109/ICRA.2012.6225257.

[23] Alonzo Church (1962): *Logic, arithmetic and automata*. In: Proceedings of the international congress of mathematicians, pp. 23–35.

[24] Niklas Eén, Alexander Legg, Nina Narodytska & Leonid Ryzhyk (2015): *SAT-Based Strategy Extraction in Reachability Games*. In: AAAI, AAAI Press, pp. 3738–3745. Available at `http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9683`.

[25] Rüdiger Ehlers (2011): *Unbeast: Symbolic Bounded Synthesis*. In: TACAS, LNCS 6605, Springer, pp. 272–275, doi:10.1007/978-3-642-19835-9_25.

[26] Rüdiger Ehlers (2012): *Symbolic bounded synthesis*. Formal Methods in System Design 40(2), pp. 232–262, doi:10.1007/s10703-011-0137-x.

[27] Emmanuel Filiot: *Acacia+*. Available at `http://lit2.ulb.ac.be/acaciaplus/`.

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

[29] Emmanuel Filiot, Naiyong Jin & Jean-François Raskin (2011): *Antichains and compositional algorithms for LTL synthesis*. Formal Methods in System Design 39(3), pp. 261–296.

[30] Emmanuel Filiot, Naiyong Jin & Jean-François Raskin (2013): *Exploiting structure in LTL synthesis*. STTT 15(5-6), pp. 541–561, doi:10.1007/s10009-012-0222-5.

[31] Bernd Finkbeiner & Swen Jacobs (2012): *Lazy Synthesis*. In: VMCAI, LNCS 7148, Springer, pp. 219–234, doi:10.1007/978-3-642-27940-9_15.

[32] Bernd Finkbeiner, Markus N. Rabe & César Sánchez (2015): *Algorithms for Model Checking HyperLTL and HyperCTL**. In: CAV, LNCS 9206, Springer, pp. 30–48, doi:10.1007/978-3-319-21690-4_3.

[33] Bernd Finkbeiner & Sven Schewe (2013): *Bounded synthesis*. STTT 15(5-6), pp. 519–539, doi:10.1007/s10009-012-0228-z.

[34] Swen Jacobs: *Extended AIGER Format for Synthesis*. In *arXiv:1405.5793*, May 2014.

[35] Swen Jacobs, Roderick Bloem, Romain Brenguier, Rüdiger Ehlers, Timotheus Hell, Robert Könighofer, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup & Adam Walker (2016): *The first reactive synthesis competition (SYNTCOMP 2014)*. STTT, doi:10.1007/s10009-016-0416-3.

[36] Swen Jacobs, Roderick Bloem, Romain Brenguier, Robert Könighofer, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup & Adam Walker (2016): *The Second Reactive Synthesis Competition (SYNTCOMP 2015).* In: *SYNT 2015*, *EPTCS* 202, pp. 27–57, doi:10.4204/EPTCS.202.4.

[37] Swen Jacobs & Felix Klein (2016): *A High-Level LTL Synthesis Format: TLSF v1.1 (Extended Version). CoRR* abs/1604.02284. Available at `http://arxiv.org/abs/1604.02284`.

[38] Matti Järvisalo, Daniel Le Berre, Olivier Roussel & Laurent Simon (2012): *The International SAT Solver Competitions*. *AI Magazine* 33(1).

[39] Barbara Jobstmann & Roderick Bloem: *Lily — a LInear Logic sYnthesizer*. `http://www.iaik.tugraz.at/content/research/design_verification/lily/`.

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

[41] Ayrat Khalimov (2016): *Specification Format for Reactive Synthesis Problems*. In: *SYNT*, *EPTCS* 202, pp. 112–119, doi:10.4204/EPTCS.202.8.

[42] Hadas Kress-Gazit, Georgios E. Fainekos & George J. Pappas (2009): *Temporal-Logic-Based Reactive Mission and Motion Planning*. *IEEE Transactions on Robotics* 25(6), pp. 1370–1381, doi:10.1109/TRO.2009.2030225.

[43] Jun Liu, Necmiye Ozay, Ufuk Topcu & Richard M. Murray (2013): *Synthesis of Reactive Switching Protocols From Temporal Logic Specifications*. *IEEE Trans. Automat. Contr.* 58(7), pp. 1771–1785.

[44] P. Madhusudan & P. S. Thiagarajan (2001): *Distributed Controller Synthesis for Local Specifications*. In: *ICALP*, *LNCS* 2076, Springer, pp. 396–407, doi:10.1007/3-540-48224-5_33.

[45] P. Madhusudan & P. S. Thiagarajan (2002): *A Decidable Class of Asynchronous Distributed Controllers*. In: *CONCUR*, *LNCS* 2421, Springer, pp. 145–160, doi:10.1007/3-540-45694-5_11.

[46] Andreas Morgenstern & Klaus Schneider (2011): *A LTL Fragment for GR(1)-Synthesis*. In: *iWIGP*, *EPTCS* 50, pp. 33–45.

[47] Nina Narodytska, Alexander Legg, Fahiem Bacchus, Leonid Ryzhyk & Adam Walker (2014): *Solving Games without Controllable Predecessor*. In: *CAV*, *LNCS* 8559, Springer, pp. 533–540, doi:10.1007/978-3-319-08867-9_35.

[48] Amir Pnueli (1977): *The Temporal Logic of Programs*. In: *FOCS*, IEEE Computer Society, pp. 46–57, doi:10.1109/SFCS.1977.32.

[49] Amir Pnueli & Roni Rosner (1989): *On the Synthesis of a Reactive Module*. In: *POPL*, ACM Press, pp. 179–190, doi:10.1145/75277.75293.

[50] Amir Pnueli & Roni Rosner (1990): *Distributed Reactive Systems Are Hard to Synthesize*. In: *Foundations of Computer Science (FOCS'90)*, IEEE Computer Society, pp. 746–757, doi:10.1109/FSCS.1990.89597.

[51] Michael O. Rabin (1972): *Automata on Infinite Objects and Church's Problem*. *Amer. Math. Soc.* 13.

[52] Leonid Ryzhyk, Adam Walker, John Keys, Alexander Legg, Arun Raghunath, Michael Stumm & Mona Vij (2014): *User-Guided Device Driver Synthesis*. In: *OSDI*, USENIX Association, pp. 661–676. Available at `https://www.usenix.org/conference/osdi14/technical-sessions/presentation/ryzhyk`.

[53] Saqib Sohail & Fabio Somenzi (2013): *Safety first: a two-stage algorithm for the synthesis of reactive systems*. *STTT* 15(5-6), pp. 433–454, doi:10.1007/s10009-012-0224-3.

[54] Aaron Stump, Geoff Sutcliffe & Cesare Tinelli (2014): *StarExec: A Cross-Community Infrastructure for Logic Solving*. In: *IJCAR*, *LNCS* 8562, Springer, pp. 367–373, doi:10.1007/978-3-319-08587-6_28.

[55] Geoff Sutcliffe & Christian B. Suttner (2006): *The state of CASC*. *AI Commun.* 19(1), pp. 35–48.

[56] Leander Tentrup (2016): *Solving QBF by Abstraction*. *CoRR* abs/1604.06752. Available at `http://arxiv.org/abs/1604.06752`.

[57] Wolfgang Thomas (1995): *On the Synthesis of Strategies in Infinite Games*. In: *STACS*, pp. 1–13, doi:10.1007/3-540-59042-0_57.

[58] Cong Tian, Jun Song, Zhenhua Duan & Zhao Duan (2015): *LtlNfBa: Making LTL Translation More Practical*. In: *SOFL+MSVL*, *LNCS* 9559, Springer, pp. 179–194, doi:10.1007/978-3-319-31220-0_13.

[59] Adam Walker & Leonid Ryzhyk (2014): *Predicate abstraction for reactive synthesis*. In: *FMCAD*, IEEE, pp. 219–226, doi:10.1109/FMCAD.2014.6987617.

[60] Martin Zimmermann (2013): *Optimal bounds in parametric LTL games*. *Theor. Comput. Sci.* 493, pp. 30–45, doi:10.1016/j.tcs.2012.07.039.