SMT-Based CPS Parameter Synthesis*

(Tool Presentation)

Heinz Riener[1], Robert Könighofer[2], Goerschwin Fey[1], and Roderick Bloem[2]

[1] Institute of Space Systems, German Aerospace Center, Bremen, German
heinz.riener@dlr.de, goerschwin.fey@dlr.de
[2] IAIK, Graz University of Technology, Graz, Austria
robert.koenighofer@iaik.tugraz.at, roderick.bloem@iaik.tugraz.at

Abstract

We present a simple, yet flexible parameter synthesis approach for Cyber-Physical Systems (CPS). The user defines the behavior of a CPS, a set of (un)safe states, and a generic template for an invariant using Satisfiability Modulo Theories (SMT) formulas. Counterexample-Guided Inductive Synthesis (CEGIS) is then used to compute values for open parameters and a concrete invariant to prove that all unsafe states are unreachable. We present a proof-of-concept tool, optimizations, and first experiments.

1 Introduction

Cyber-Physical Systems (CPS) [12] integrate computing capabilities with monitoring and control of entities in the physical world. This interaction quickly results in a high complexity, even for small systems. Subtle issues are easy to overlook by manual inspection and simulation, calling for more rigorous (formal) methods. Methods that cannot only verify a given CPS but also assist during its design are particularly appealing.

The crux in designing specific aspects of a CPS is often in finding suitable values for important design parameters such that given requirements are satisfied. As an example, think of a controller to keep some temperature within a certain range. A two-position controller (switching the heating on and off if certain temperature thresholds are exceeded) is easy to implement. The difficulty lies in finding parameter values like the polling interval, the temperature thresholds, the heating rate, etc. Especially in the presence of uncontrollable inputs or events from the environment, corner cases can easily be overlooked when defining such parameters.

We thus propose a parameter synthesis approach for CPS that is based on formal methods. The user characterizes the CPS behavior using Satisfiability Modulo Theories (SMT) formulas. This includes a definition of how the CPS evolves over time, and the possible initial states. As specification, the user defines unsafe states that must not be visited. Finally, the user provides a template for an invariant to prove correctness. All these definitions can reference parameters for which the value is yet unknown. In particular, parameters can be used in the invariant to specify only its abstract shape. E.g., for our temperature control example, the unsafe states could be characterized by having a temperature $t > 25°C$ or $t < 18°C$, the invariant could be that the temperature resides within some interval $[k_l, k_u]$, where k_l and k_u are open parameters, and other design choices like the polling interval and the heating rate are open parameters, too.

Our approach automatically combines these ingredients into a compact correctness formula. To increase the chances for proving unbounded correctness with the given invariant template, we define a notion of n-step inductiveness, requiring the invariant not to hold always but only

*This work was supported in part by the European Commission through the project IMMORTAL (644905) and by the Austrian Science Fund (FWF) through the research network RiSE (S11406-N23).

every $\leq n$ transitions (while only safe states can be visited in the meantime). Counterexample-Guided Inductive Synthesis (CEGIS) [16], a technique for parameter synthesis in software, is then applied to break the parameter synthesis problem into simple (unquantified) queries for an SMT solver. We also propose optimizations to speed up convergence in our setting. There are three possible outcomes: (1) the tool produces parameter values such that the CPS cannot reach any unsafe state, (2) the tool reports that no such values exist, or (3) the computation is aborted (e.g., because of a timeout or an incompleteness of the solver for the theories used).

Our approach is flexible. The user can pick the SMT solver and theory that is most suitable for the problem at hand. For simple problems, linear arithmetic over the reals may suffice, and a solver tailored towards such theories may yield the best performance. For highly non-linear CPS, specialized SMT solvers like dReal [10] can be used. Our approach can also benefit from user insights. If the tool fails to find a solution, the user can improve the invariant, the definition of the CPS behavior, etc. Finally, our approach is simple, which makes it easy to extend and to customize. SMT provides a rich tool infrastructure for such extensions, e.g., optimizing solvers [4, 14] to minimize the cost of the solution regarding some quantitative metric.

Related work. Frehse et al. [9] present a counterexample-guided approach to parameter synthesis, but consider only linear hybrid automata. It computes an underapproximation of the set of good parameter configurations. Bogomolov et al. [5] compute a parameter region for multiaffine hybrid automata using abstraction with linear hybrid automata. Cimatti et al. [7] compute all good parameter configurations precisely for CPS that are modeled as symbolic transition systems with linear constraints. Our approach computes only one parameter configuration, which is potentially more efficient. Since we are synthesizing parameters anyway, we also synthesize an invariant (from a user-given template) along the way, which enables reasoning about unbounded correctness using succinct formulas. We apply CEGIS [16] with some optimizations to synthesize parameters for CPS, using an SMT solver as a black box. We only require that the solver can compute satisfying assignments. Cheng et al. [6] present a more sophisticated approach (like "CEGIS on steroids") to solve formulas of the form $\exists x : \forall y : \varphi$, utilizing more advanced solver features. We can also use [6] instead of CEGIS in our approach. Cheng et al. [6] also present examples, demonstrating that many design problems for CPS reduce to $\exists\forall$-formulas. Some examples also use (templates for) invariants to reason about CPS correctness. With our notion of n-step inductive invariants, we generalize this concept. We are not aware of existing work using such invariants, but there are similarities to k-induction [15].

Contributions. In summary, the main contributions of this paper are (1) to provide a flexible flow for CPS parameter synthesis based on existing algorithms and engines, (2) our notion of n-step inductive invariants to reason about unbounded CPS correctness, and (3) a proof-of-concept tool with optimizations and heuristics to improve the performance in practice.

Outline. The next section presents our approach from a user's perspective and from an algorithmic perspective. Section 3 discusses our tool and first experiments. Section 4 concludes.

2 Parameter Synthesis Approach

2.1 A User's Perspective

The SMT-LIB [2] standard defines a common language for SMT solvers. To maximize flexibility, our approach operates on parameter synthesis problems that are defined directly in this format.

The user defines two datatypes, one to represent a state, and one to represent input. The state Q is formed by a vector of state variables $\overline{q} = (q_1, \ldots, q_a)$. Individual state variables can be of different type (e.g., one variable may be a Boolean flag, another one a real value to

represent time or some other physical quantity, yet another one an enumeration type with 8 options). Similarly, the input I is defined using a vector of input variables $\overline{i} = (i_1, \ldots, i_b)$. The user can also define constants (of different type) to represent parameters of the CPS. We denote the parameter variables by $\overline{k} = (k_1, \ldots, k_c)$ and the set of all parameter valuations by K.

The user defines the dynamics of the CPS via a transition function $T : Q \times I \times K \to Q$, which defines the next state $q' = T(q, i, k)$ based on current state $q \in Q$, the input $i \in I$, and the parameter values $k \in K$. Note that T defines q' uniquely based on q, i and k. Uncertainties about the next state need to be "externalized" by introducing additional input variables. Also note that T does not restrict the formalism to discrete-time systems. Continuous time can be modeled, e.g., by having one real-valued variable to represent the elapsed time (or several such variables to represent multiple clocks). An input variable may define how much real time elapses by an application of T. Thus, T can be thought of as defining possible sequences of visible states rather than imposing discrete time steps.

The set of possible initial states of the CPS is defined with a function $\mathsf{init} : Q \times K \to \mathbb{B}$. Similarly, $\mathsf{safe} : Q \times K \to \mathbb{B}$ defines the safe states, and $\mathsf{inv} : Q \times K \to \mathbb{B}$ characterizes the states that satisfy a user-given invariant. All these functions can be defined in SMT-LIB syntax in a rather straightforward way. Also note that all these functions can reference the parameters.

In order to formalize what it means for a CPS to be correct, we inductively define the set $\mathsf{reach}(k) \subseteq Q$ of states that are reachable with parameter values $k \in K$ as follows:

- All initial states are reachable, i.e., $\forall q \in Q : \mathsf{init}(q, k) \to q \in \mathsf{reach}(k)$.
- If a state is reachable, then all of its possible successor states are reachable, i.e., $\forall q \in Q : \forall i \in I : q \in \mathsf{reach}(k) \to T(q, i, k) \in \mathsf{reach}(k)$.
- No other state is reachable.

Our tool attempts to compute parameter values $k \in K$ such that all reachable states are safe, i.e., $\forall q \in Q : q \in \mathsf{reach}(k) \to \mathsf{safe}(q, k)$. The next section explains how this works.

2.2 Under the Hood

Our parameter synthesis approach consists of two steps. First, we construct a correctness formula $\mathsf{correct} : I' \times K \to \mathbb{B}$ such that $\mathsf{correct}(i', k)$ evaluates to true (only) if the CPS with parameter values $k \in K$ is correct when fed with input (sequence) $i' \in I'$. Second, we compute parameter values $k \in K$ such that $\forall i' \in I' : \mathsf{correct}(i', k)$ holds. Note that this inherently involves handling alternating quantifiers ($\exists k : \forall i' : \ldots$). Since SMT solvers have their strengths in solving unquantified formulas, we apply CEGIS [16] to split the problem into a sequence of unquantified formulas for an SMT solver.

Defining the correctness formula. A common approach to define a correctness formula from a transition function T is known as Bounded Model Checking (BMC) [3]: T is unfolded for an increasing number of steps, while asserting that no unsafe state can be visited. This approach is not only common for reactive systems but also applied to CPS (see, e.g., [11]). The drawbacks are that (1) BMC only gives bounded correctness guarantees, and (2) can result in large correctness formulas (containing many copies of T). Thus, we rather follow an invariant-based approach. By induction, a CPS cannot visit an unsafe state (and is thus correct) if

1. all initial states satisfy the invariant,
2. from a state that satisfies the invariant, the next state necessarily satisfies the invariant as well, and
3. all states that satisfy the invariant are also safe.

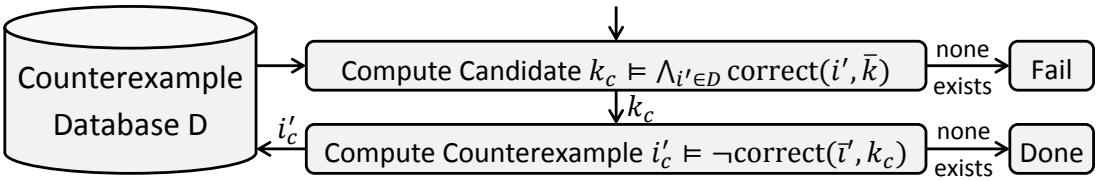

Figure 1: Counterexample-Guided Inductive Synthesis (CEGIS) [16].

In our setting, this insight can be used to define a correctness formula as $\mathsf{correct}_1(q, i, k) =$

$$\Big(\mathsf{init}(q,k) \to \mathsf{inv}(q,k)\Big) \wedge \Big(\mathsf{inv}(q,k) \to \mathsf{inv}(T(q,i,k),k)\Big) \wedge \Big(\mathsf{inv}(q,k) \to \mathsf{safe}(q,k)\Big).$$

If we find parameter values $k \in K$ such that $\forall q \in Q, i \in I : \mathsf{correct}_1(q,i,k)$ holds, then these parameter values induce a correct system. We can thus define $\mathsf{correct}$ from $\mathsf{correct}_1$ by setting $I' = Q \times I$, i.e., treat states as if they were uncontrollable inputs.

n-step inductiveness. The formula $\mathsf{correct}_1$ from the previous paragraph requires the invariant to hold *always*. We call such an invariant 1-*step inductive* (because it holds again after at most 1 step of T). If the unsafe states are unreachable, it is always possible to find such a 1-step inductive invariant (the set of reachable states can be used). Yet, the user provided invariant template inv may be too restrictive to express such an invariant. We discovered such cases in our experiments.[1] There are two options: (1) the user can improve the invariant template, and (2) we can relax the requirement that the invariant must hold in *all* steps.

For the second option, we define a notion of n-*step inductiveness*. It requires that, whenever the invariant is satisfied, it will be satisfied again after at most n steps of T, no matter what the inputs are. Additionally, all states that can be reached in the meantime must be safe. We can formalize this requirement by defining $\mathsf{correct}_n(q_0, i_1, \ldots, i_n, k) =$

$$\Big(\mathsf{init}(q_0,k) \to \mathsf{inv}(q_0,k)\Big) \wedge \Big(\mathsf{inv}(q_0,k) \to \mathsf{safe}(q_0,k)\Big) \wedge \Big(\mathsf{inv}(q_0,k) \to \bigvee_{j=1}^{n} \mathsf{inv}(q_j,k) \wedge \bigwedge_{l=1}^{j-1} \mathsf{safe}(q_l,k)\Big)$$

where q_j is an abbreviation for $T(q_{j-1}, i_j, k)$ for all $j > 0$. Similar to before, we want to find parameter values $k \in K$ such that $\forall q_0 \in Q : \forall i_1, \ldots, i_n \in I : \mathsf{correct}_n(q_0, i_1, \ldots, i_n, k)$ holds. We can thus define $\mathsf{correct}$ from $\mathsf{correct}_n$ by setting $I' = Q \times I^n$.

CEGIS. Now that we have defined $\mathsf{correct} : I' \times K \to \mathbb{B}$, we can turn to computing parameters $k \in K$ such that $\forall i' \in I' : \mathsf{correct}(i', k)$ holds. We use Counterexample-Guided Inductive Synthesis (CEGIS) [16], a technique that has been invented for parameter synthesis in software. The basic idea of CEGIS is to refine candidate values for the parameters iteratively based on counterexamples until a correct solution is found. This is illustrated in Figure 1. There is a database $D \subseteq I'$ of concrete input values, which is initially empty. In the first step of a loop, parameter candidates k_c are computed such that $\bigwedge_{i' \in D} \mathsf{correct}(i', k_c)$ holds, i.e., the CPS is correct for all inputs from the database D. In our case, an SMT solver is used to compute such values k_c as satisfying assignment to the formula $\bigwedge_{i' \in D} \mathsf{correct}(i', \bar{k})$, where $\bar{k}$ are the

[1] E.g., for our temperature control example, an invariant that only requires the temperature t to reside within some interval $[k_l, k_u]$ can never be 1-step inductive: No matter how k_u is chosen, there exists a state with t close to k_u and heating on such that k_u is exceeded after a step of T. However, after violating the invariant temporarily, some application of T will turn the heating off eventually, so t will fall back into the interval $[k_l, k_u]$ after some number of steps. Thus, $t \in [k_l, k_u]$ may still be an n-step inductive invariant for some k_l, k_u.

parameter variables. If no such values exist (the formula is unsatisfiable), then this means that the parameter synthesis problem has no solution, so the loop aborts. If values k_c have been found, the next step in the loop is to check whether they result in a correct CPS behavior for *all* input values $i' \in I'$ (and not just those from D). In order to answer this question, we use an SMT solver to check if $\neg\mathsf{correct}(\vec{i}', k_c)$ is satisfiable (where $\vec{i}'$ are input variables that are left open), i.e., if there exists an input for which the CPS is not correct. If the formula is unsatisfiable, the values k_c are a solution for the parameter synthesis problem and the algorithm terminates. If the formula is satisfiable, we compute a satisfying assignment i'_c and add it to the database D. This has the effect that the candidate parameters computed in the next iteration are "better" in the sense that they work for the input i'_c as well. The loop is not guaranteed to terminate (unless I' or K is finite). Once concrete parameter variables k_c have been found, the loop can be continued with the additional constraint $\overline{k} \neq k_c$ in order to find other solutions.

We apply the CEGIS loop on $\mathsf{correct}_n$ with increasing values of n until a solution is found. User-given bounds on the maximum value of n, the number of iterations in the CEGIS loop, etc., can be imposed to ensure termination. Setting a timeout may be even more convenient. If the SMT solver is incomplete for the theories being used (e.g., because the theories are undecidable even for the unquantified case), it can happen that the SMT solver returns `unknown` as a result. In such cases, we present the latest parameter candidate k_c to the user and abort.

3 Implementation and First Experiments

We implemented our parameter synthesis approach as a proof-of-concept tool using Python 3 and conducted first experiments. All experiments were performed on a quad-core Intel® Core™ i5-2520M CPU with 2.50GHz and 8GB RAM. Z3 [8] 4.3.1 was used as SMT solver. Our tool as well as the input files for reproducing our experiments are available for download[2].

Implementation. The proof-of-concept tool implements the CEGIS synthesis loop shown in Figure 1. In contrast to CEGIS for parameter synthesis for software, in our case the domains of continuous variables are infinite such that the termination of the CEGIS loop is in general not guaranteed. We implemented a few simple heuristics to improve convergence. The first two heuristics were designed such that our benchmarks terminate. The third heuristic was added to improve the performance.

1. *Counterexample randomization*: To avoid generating too similar counterexamples, our proof-of-concept tool attempts to randomize every second counterexample. In an iterative loop, for each value of the counterexample, a random value of the same type is generated and substituted. If the adapted counterexample still violates the correctness check, the randomized value is kept. Otherwise, it is rejected.
2. *Restart strategy*: Inspired by the implementation of today's CDCL SAT solvers, we implemented a simple restart strategy. When a restart happens, all counterexamples are removed from the database and the CEGIS synthesis loop starts from the beginning without a priori knowledge. After each restart, the period of the restart is increased.
3. *Demand for progress*: Given two subsequent values k_a and k_b of the same parameter, we measure their progress by $\mathsf{progress}(k_a, k_b) = \|k_a - k_b\|$. This measure is used to restart the synthesis procedure when the CEGIS loop gets stuck by producing similar counterexamples, but counterexample randomization is not effective. In each iteration, for the last pair of parameter values k_{c-1} and k_c, the progress value $\mathsf{progress}(k_{c-1}, k_c)$ is computed. If

[2] https://github.com/hriener/parsyn-cegis

Benchmark	Discrete State Space	Continuous State Space	
	#Locations	#Reals	#Integers
`thermal1-s2-safe`	2	1	0
`thermal2-s3-safe`	3	1	2
`water-s3-unsafe`	3	1	0
`water-s4-safe`	4	1	1

Table 1: Benchmark characteristics.

the progress value repeatedly falls below a fixed progress threshold δ, e.g., more than 10 times, a restart is initiated.

Experiments. We used our proof-of-concept tool to concretize the parameter values of invariant templates for a selected set of benchmarks. The counterexample randomization and the restart strategy were enabled for all experiments such that the CEGIS loop always terminates for the considered benchmarks. The restart threshold was set to 16 initially and incremented by 16 on every restart. For the progress heuristic, we repeated the experiments twice with enabled and disabled heuristic, respectively, to analyze the speed-up. The fixed progress threshold $\delta = 10^{-7}$ was configured.

Benchmarks. As benchmarks, we considered simple examples of hybrid systems from the literature as SMT instances in the format described in Section 2.1. The examples include two different temperature control systems (similar to [1]) and two version of a simple water tank (similar to [13]). An overview of the benchmark characteristics is given in Table 1. The table lists the number of discrete locations as well as the numbers and types of continuous variables by the benchmark.

Results. Table 2 lists the averaged results over 100 runs for the benchmarks with different values for n in $\mathsf{correct}_n$. The table is built as follows: the first two columns name the benchmark and show the number of parameters to be synthesized. The next three columns show the average number of SMT solver calls, generated counterexamples, and whether parameter values were synthesized, respectively. The number of SMT solver calls includes the calls for parameter synthesis, correctness checking as well as counterexample randomization. All results are as expected, i.e., if we report 'Yes' in the column 'Found', parameter values for the invariant template could be synthesized. Otherwise, if we report 'No', no such parameter values exist. The last four columns show the average amount of run-time required for CEGIS synthesis, where t_P is the time required for finding parameters, t_C is the time for correctness checking, t_R is the time for counterexample randomization and Σ is the total time. For the upper half of the table, the progress heuristic was disabled, and for the lower half enabled.

Discussion. The execution times for the benchmarks in our experiments are very low, even though we do not use any sophisticated features of the underlying solver. Comparing the upper and the lower half of the table shows that our progress heuristic gives a solid speed-up for the `water-s3-unsafe` benchmark while there is only a minor effect on the performance of the other cases. Note that the total solving time does not correlate with the number of steps (n), since the time per solver call can vary.

Benchmark	$\lvert\overline{\mathbf{k}}\rvert$	n	#SMT	#CEX	Found	Time			
						$\mathbf{t_P}$ [s]	$\mathbf{t_C}$ [s]	$\mathbf{t_R}$ [s]	$\mathbf{\Sigma}$ [s]
thermal1-s2-safe	2	1	53.56	21.12	No	0.19	0.19	0.09	0.49
thermal1-s2-safe	2	2	34.69	14.16	Yes	0.12	0.15	0.06	0.33
thermal2-s3-safe	4	1	179.37	51.26	No	0.78	0.74	1.16	2.72
thermal2-s3-safe	4	2	528.28	150.96	No	4.22	3.43	6.50	14.29
thermal2-s3-safe	4	3	151.43	43.93	Yes	1.07	1.04	1.64	3.80
water-s3-unsafe	2	1	1663.14	664.92	No	57.96	31.42	28.00	117.92
water-s3-unsafe	2	2	25.88	10.07	No	0.13	0.14	0.06	0.34
water-s4-safe	2	1	32.00	11.00	Yes	0.09	0.11	0.09	0.30
thermal1-s2-safe	2	1	55.48	21.89	No	0.20	0.19	0.09	0.50
thermal1-s2-safe	2	2	35.41	14.46	Yes	0.12	0.15	0.06	0.34
thermal2-s3-safe	4	1	235.83	67.38	No	1.06	1.01	1.59	3.74
thermal2-s3-safe	4	2	580.95	166.02	No	4.67	3.82	7.14	15.85
thermal2-s3-safe	4	3	149.00	43.27	Yes	1.11	1.07	1.67	3.91
water-s3-unsafe	2	1	908.95	363.26	No	5.59	5.39	2.83	14.07
water-s3-unsafe	2	2	28.23	11.02	No	0.14	0.16	0.06	0.37
water-s4-safe	2	1	32.00	11.00	Yes	0.09	0.11	0.09	0.30

Table 2: Parameter synthesis.

4 Summary and Conclusion

We presented a simple and flexible parameter synthesis approach for CPS. The user defines the behavior of the CPS, a set of (un)safe states, and a generic template for an inductive invariant in SMT. Our approach then synthesizes the open parameters using CEGIS. We presented a proof-of-concept tool, optimizations, and promising first experiments.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *TCS*, 138(1):3–34, 1995.

[2] C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa, 2015. Available at `www.SMT-LIB.org`.

[3] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS'99*, LNCS 1579, pages 193–207. Springer, 1999.

[4] N. Bjørner, A. Phan, and L. Fleckenstein. νz - an optimizing SMT solver. In *TACAS'15*, LNCS 9035, pages 194–199. Springer, 2015.

[5] S. Bogomolov, C. Schilling, E. Bartocci, G. Batt, H. Kong, and R. Grosu. Abstraction-based parameter synthesis for multiaffine systems. In *HVC'15*, LNCS 9434, pages 19–35. Springer, 2015.

[6] C.-H. Cheng, N. Shankar, H. Ruess, and S. Bensalem. EFSMT: A logical framework for cyber-physical systems. *CoRR*, abs/1306.3456, 2013.

[7] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Parameter synthesis with IC3. In *FMCAD'13*, pages 165–168. IEEE, 2013.

[8] L. M. de Moura and N. Bjørner. Z3: an efficient SMT solver. In *TACAS'08*, LNCS 4963, pages 337–340. Springer, 2008.

[9] G. Frehse, S. K. Jha, and B. H. Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In *HSCC'08*, LNCS 4981, pages 187–200. Springer, 2008.

[10] S. Gao, S. Kong, and E. M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *CADE-24*, LNCS 7898, pages 208–214. Springer, 2013.

[11] S. Kong, S. Gao, W. Chen, and E. M. Clarke. dReach: δ-reachability analysis for hybrid systems. In *TACAS'15*, LNCS 9035, pages 200–205. Springer, 2015.

[12] E. A. Lee and S. A. Seshia. *Introduction to Embedded Systems — A Cyber-Physical Systems Approach*. LeeSeshia.org, second edition, 2015.

[13] A. Platzer. *Logical Analysis of Hybrid Systems - Proving Theorems for Complex Dynamics*. Springer, 2010.

[14] R. Sebastiani and P. Trentin. Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions. In *TACAS'15*, LNCS 9035, pages 335–349. Springer, 2015.

[15] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In *FMCAD'00*, LNCS 1954, pages 108–125. Springer, 2000.

[16] A. Solar-Lezama. Program sketching. *STTT*, 15(5-6):475–495, 2013.